高等学校计算机专业教材精选·网络与通信技术

Web开发技术实用教程

陈轶 主编

肖建 王炜立 副主编

清华大学出版社

高等学校计算机专业教材精选·网络与通信技术

Web开发技术实用教程

陈 铁 主 编
肖 建 王炜立 副主编

清华大学出版社
北 京

内 容 简 介

本书立足于Web技术发展状况和特点，从读者学习当前主流Web技术出发，系统地介绍以JSP技术为主体的Web开发应用技术。

本书分成14章，内容安排合理，实用性强。涵盖了当前Web开发技术的主要内容，具体涉及Web技术基础、HTML技术基础、XHTML技术、CSS技术、客户端脚本语言、JSP开发的Java语言基础、JSP的开发体系和环境配置、JSP的主要内置对象、JSP的其他内置对象、JSP的文件操作、JSP的JavaBean编程、JSP的Servlet编程、JSP访问Web数据库、XML技术以及Web的综合应用。并介绍了无线标记语言和无线标记脚本语言，利用它们开发无线Web应用。在每一章都提供了具有现实意义的实例，帮助读者了解和掌握相关技术。

本书可以作为高等学校计算机及相关专业本科生、专科生、高职生和各类成人教育学院的Web程序设计、Web技术、网页设计和JSP技术课程教材，也可供相关技术人员使用。

图书在版编目（CIP）数据

Web开发技术实用教程/陈铁主编．—北京：清华大学出版社，2008.9
（高等学校计算机专业教材精选·网络与通信技术）
ISBN 978-7-302-17435-6

Ⅰ．W…　Ⅱ．陈…　Ⅲ．主页制作－程序设计－高等学校－教材　Ⅳ．TP393.092

中国版本图书馆CIP数据核字(2008)第125632号

责任编辑：汪汉友
责任校对：梁　毅
责任印制：孟凡玉

出版发行：清华大学出版社　　**地　　址：**北京清华大学学研大厦A座
http://www.tup.com.cn　　**邮　　编：**100084
社　总　机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京密云胶印厂
装 订 者：三河市溧源装订厂
经　　销：全国新华书店
开　　本：185×260　　**印　张：**27　　**字　　数：**635千字
版　　次：2008年9月第1版　　**印　　次：**2008年9月第1次印刷
印　　数：1～4000
定　　价：39.00元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。
联系电话：010-62770177转3103　　产品编号：028358-01

出版说明

我国高等学校计算机教育近年来迅猛发展，应用所学计算机知识解决实际问题，已经成为当代大学生的必备能力。

时代的进步与社会的发展对高等学校计算机教育的质量提出了更高、更新的要求。现在，很多高等学校都在积极探索符合自身特点的教学模式，涌现出一大批非常优秀的精品课程。

为了适应社会的需求，满足计算机教育的发展需要，清华大学出版社在进行了大量调查研究的基础上，组织编写了《高等学校计算机专业教材精选》。本套教材从全国各高校的优秀计算机教材中精挑细选了一批很有代表性且特色鲜明的计算机精品教材，把作者们对各自所授计算机课程的独特理解和先进经验推荐给全国师生。

本系列教材特点如下。

(1) 编写目的明确。本套教材主要面向广大高校的计算机专业学生，使学生通过本套教材，学习计算机科学与技术方面的基本理论和基本知识，接受应用计算机解决实际问题的基本训练。

(2) 注重编写理念。本套教材作者群为各校相应课程的主讲，有一定经验积累，且编写思路清晰，有独特的教学思路和指导思想，其教学经验具有推广价值。本套教材中不乏各类精品课配套教材，并力图把不同学校的教学特点反映到每本教材中。

(3) 理论知识与实践相结合。本套教材贯彻从实践中来到实践中去的原则，书中的许多必须掌握的理论都将结合实例来讲，同时注重培养学生分析、解决问题的能力，满足社会用人要求。

(4) 易教易用，合理适当。本套教材编写时注意结合教学实际的课时数，把握教材的篇幅。同时，对一些知识点按教育部教学指导委员会的最新精神进行合理取舍与难易控制。

(5) 注重教材的立体化配套。大多数教材都将配套教师用课件、习题及其解答，学生上机实验指导、教学网站等辅助教学资源，方便教学。

随着本套教材陆续出版，相信能够得到广大读者的认可和支持，为我国计算机教材建设及计算机教学水平的提高，为计算机教育事业的发展做出应有的贡献。

清华大学出版社

前　　言

伴随着 Internet 应用领域的不断扩展和 Web 技术的突飞猛进，Web 应用已经成为现代社会不可或缺的一部分。“Web 开发技术”以及相关课程得到许多学校的关注。为了适应计算机教学发展趋势，有必要编写一本符合当前 Web 技术发展趋势和教学现状的教材，帮助广大学生了解和掌握 Web 的主流技术。

这本教材是作者总结多年 Web 项目开发以及 Web 教学实践的经验，结合相关的技术资料编写而成。本书从程序设计角度出发，紧密结合 Web 开发技术特点和高校 Web 开发课程的教学大纲，力求覆盖当前主流 Web 开发中使用的常用技术，涉及了当前 Web 应用中常见的客户端开发、服务器端开发技术，侧重介绍了服务器端的编程常用的 JSP 技术的基本内容。此外，本书还引入了 WAP 2.0 编程，引导读者进入无线 Web 开发领域，有效填补当前同类教材在该领域的空白。

本书分成 14 章，涵盖了当前 Web 开发技术中的 Web 技术基础、Web 前台技术和后台技术三部分部分内容，具体包括如下内容。

第 1 部分：Web 技术基础

第 1 章了解 Web 开发技术基础，具体设计了 Web 的基本概念、Web 发展状况、工作原理、工作模式和关键技术。

第 2 部分：Web 客户端开发技术

第 2 章从脚本语言发展的角度出发，依次介绍了 HTML 的基本概念以及常见的基本标签和 XHTML 主要内容。从而详细地解释了 XHTML 相较于 HTML 的特点，并通过具体的应用实例展示了 XHTML 的优势。

第 3 章介绍了 CSS 基本语法以及常见的基本属性。通过对 CSS 介绍，了解如何用 CSS 创建生动的网页外观。

第 4 章主要对 JavaScript 脚本语言展开介绍，具体涉及的内容有 JavaScript 脚本语言的基本语法、JavaScript 的控制流程、JavaScript 的函数、JavaScript 的事件处理和 JavaScript 的内置对象。通过对它们的介绍，为进一步学习 Ajax 打下坚实的基础。

第 5 章主要介绍了 XML 在客户端的应用，包括了 XML 基础、XML 的验证机制 DTD 和 XML Schema、XML 的 CSS 显示，以及 XPATH 基础和 XSLT 转换 XML。

第 6 章介绍 WAP 2.0 无线应用协议，具体涉及的内容有 WML 1.3、WML 2.0、XHTML Basic、XHTML Mobile Profile 和 WMLScript 等。通过它们引导读者了解无线终端的移动 Web 应用的开发。了解各式前台终端的 Web 页面设计，并为后续学习奠定基础。

第 3 部分：Web 服务器端开发技术

比较了常见的服务器端的常见开发语言，并侧重介绍了服务器端常用的编程语言 JSP。

第 7 章是为了能使没有任何编程经验的读者迅速进入学习状态而特定编写的。介绍了 Java 语言基础。如果具有 Java 编程经验的读者可以跳过这一章。

第 8 章介绍了 JSP 的工作原理和 JSP 基本语法。并介绍了 Tomcat 服务器的安装和配置，让读者能快速进入开发 JSP 应用的状态。

第 9 章介绍了 JSP 的 out、request、response、application、session、page、pageContext、config 和 exception 等 9 种内置对象，并通过具体实例对这些内置对象的应用展开详细的说明。

第 10 章介绍了 JSP 实现对文件的操作，具体内容有：File 类、JSP 的输入流和输出流、文件的相关操作如文件的写入、读取、修改以及文件的上传等。

第 11 章是 Web 开发的一个重要内容。介绍了 JDBC、JSP 访问数据的相关操作以及实现、JSP 访问数据库的常见技巧，如中文字符乱码问题的解决、分页显示和连接池的使用等内容。通过对它们的介绍，让读者可以迅速开发具有实用意义的 Web 应用。

第 12 章介绍了 JavaBean 组件技术、JavaBean 在 JSP 中的使用，以及用 JavaBean 开发具体的 JSP 应用。

第 13 章介绍了 Servlet 技术，了解 Servlet 与 JavaBean 和 JSP 技术结合开发 MVC 的 Web 应用。并通过 Servlet 实现会话管理、实现文件管理和实现数据库操作。

第 14 章主要介绍 JSP 是如何实现 XML 在服务器端的应用，是第 5 章的深入。具体介绍的内容有 JSP 生成 XML、JSP 应用 DOM API 和 SAX 2.0 API 解析 XML、JSP 应用 XML，并对 JSP 的自定义标签展开说明。本章还介绍了结合 JSP 与 XML 开发了基于 MVC 工作模式的一个技术论坛。

为了让读者较好地学习和理解 Web 开发技术，本书将每章分成两个部分：基本知识介绍和练习。在基本知识部分，针对学习过程中需要注意的知识点和一些常见的问题做了总结和归纳，帮助读者快速地对关键知识点进行了解和掌握。本书通过介绍实用案例的方式，在学习实用案例的过程中，引导读者由浅入深地学习和掌握相关知识，并能运用这些知识开发具有实用的 Web 应用。在练习部分，根据对知识点的要求不同，设计了形式多样的练习。练习中针对知识点掌握要求的不同，设计一个或多个具有实用性的实验。本书选材先进、符合 Web 开发技术的发展现况。读者可以根据自身的情况选取相关内容学习。

为了让读者方便练习，本书提供了多媒体教案以及教材介绍的实例的源代码，均可在清华大学出版社网站上下载。

本书由陈铁主持编写，其中本书第 2 章、第 5 章、第 6 章、第 12 章、第 13 章和第 14 章由陈铁编写，第 1 章、第 3 章、第 4 章和第 8 章由肖建编写，第 7 章和第 10 章由王炜立编写，第 9 章由邱桃荣和姚力文编写，第 11 章由李文编写。附录 A 由姚晓昆编写。最后由陈铁统稿。邱桃荣和姚力文两位老师是特别邀请的技术指导，他们对本书编写起到重要的作用。特别感谢杨国强研究员和陈征研究员对全书进行审阅。另外，在本书的编写过程中得到了王命延教授、杨宇仙等各位老师的大力支持和协助，在此表示衷心的感谢。

由于编者水平所限，书中难免存在错误和不足之处，恳请广大读者对本书提供宝贵意见和建议。

编　者

目　　录

第 1 章　Web 技术基础 …… 1

1.1　Internet 概述 …… 1

1.1.1　Internet 的发展 …… 1

1.1.2　Internet 的工作原理 …… 2

1.1.3　Internet 的应用领域 …… 3

1.2　Web 技术概述 …… 4

1.2.1　Web 技术简介 …… 4

1.2.2　Web 的工作模式 …… 5

小结 …… 6

练习 1 …… 6

第 2 章　HTML 和 XHTML …… 8

2.1　超文本标记语言 HTML …… 8

2.1.1　HTML 的发展历史 …… 8

2.1.2　HTML 文件的页面结构 …… 9

2.1.3　HTML 的基本标签 …… 12

2.1.4　超链接 …… 15

2.1.5　列表 …… 17

2.1.6　表格 …… 22

2.1.7　表单 …… 26

2.1.8　框架 …… 33

2.1.9　图像、文本格式及其他 …… 35

2.1.10　HTML 字符实体 …… 36

2.2　扩展超文本标记语言 XHTML …… 37

2.2.1　XHTML 文档类型定义(XHTML DTD) …… 38

2.2.2　XHTML 的语法 …… 39

2.2.3　XHTML 的应用实例 …… 41

小结 …… 42

练习 2 …… 42

第 3 章　CSS 技术 …… 44

3.1　CSS 简介 …… 44

3.2　CSS 基本语法 …… 45

3.2.1　CSS 的基本格式 …… 45

3.2.2 CSS 注释语句 …… 49
3.2.3 CSS 选择符 …… 49
3.2.4 样式表的层叠顺序 …… 50
3.3 CSS 基本属性 …… 51
3.3.1 CSS 背景属性 …… 51
3.3.2 CSS 文本属性 …… 53
3.3.3 CSS 字体属性 …… 55
3.3.4 CSS 边界属性 …… 57
3.3.5 CSS 边框属性 …… 58
3.3.6 CSS 边距属性 …… 58
3.3.7 CSS 列表属性 …… 60
3.3.8 CSS 定位属性 …… 61
小结 …… 63
练习 3 …… 63

第 4 章 客户端脚本语言 …… 65
4.1 客户端脚本语言简介 …… 65
4.1.1 客户端脚本语言的作用 …… 65
4.1.2 常见的脚本语言 …… 65
4.2 JavaScript 脚本语言概述 …… 65
4.2.1 初识 JavaScript 程序 …… 66
4.2.2 常见的数据类型 …… 67
4.2.3 变量 …… 67
4.2.4 常量 …… 68
4.2.5 运算符 …… 68
4.2.6 对象和数组 …… 70
4.3 JavaScript 的控制流程 …… 74
4.3.1 条件语句 if…else …… 74
4.3.2 选择语句 switch…case …… 74
4.3.3 计数循环语句 for …… 75
4.3.4 循环语句 for…in …… 75
4.3.5 with 语句 …… 77
4.4 JavaScript 的函数 …… 77
4.4.1 函数的定义 …… 77
4.4.2 函数的调用 …… 77
4.5 JavaScript 的事件处理 …… 79
4.5.1 事件处理 …… 79
4.5.2 事件处理方法 …… 79
4.5.3 JavaScript 预定义的事件处理器 …… 79

4.6 JavaScript 内置对象 …… 81
小结 …… 83
练习 4 …… 83

第 5 章 可扩展标记语言 XML …… 85
5.1 XML 基础 …… 85
5.1.1 什么是 XML …… 85
5.1.2 XML 的相关技术 …… 87
5.1.3 建立 XML 文件 …… 88
5.1.4 XML 的命名空间 …… 91
5.1.5 XML 的数据岛(XML Data Inland) …… 92
5.2 XML 验证机制 …… 93
5.2.1 文档类型定义 DTD …… 93
5.2.2 XML 模式定义语言(XML Schema Definition Language) …… 99
5.3 CSS 显示 XML …… 106
5.4 XSLT 转换 XML …… 108
5.4.1 XPath 基础 …… 108
5.4.2 XSLT 的基本结构 …… 112
5.4.3 用 XSLT 显示 XML …… 117
小结 …… 119
练习 5 …… 119

第 6 章 WAP 2.0 编程 …… 120
6.1 WAP 2.0 简介 …… 120
6.2 WAP 2.0 的标记语言 …… 121
6.2.1 无线标记语言 WML …… 121
6.2.2 WML 2.0 …… 129
6.2.3 XHTML Mobile Profile …… 135
6.3 WMLScript …… 138
6.3.1 WMLScript 语法基础 …… 139
6.3.2 WMLScript 常用库 …… 143
小结 …… 146
练习 6 …… 146

第 7 章 JSP 开发的 Java 语言基础 …… 148
7.1 Java 简介 …… 148
7.1.1 Java 语言特点 …… 148
7.1.2 Java 和 JavaScript 的区别 …… 149
7.2 Java 的基本语法 …… 150

7.2.1 数据类型 …… 150
7.2.2 数组 …… 152
7.2.3 常用运算 …… 155
7.2.4 控制语句 …… 156
7.3 Java 的面向对象编程基础 …… 158
7.3.1 类和对象 …… 158
7.3.2 继承性 …… 159
7.3.3 包 …… 160
7.3.4 接口 …… 161
7.3.5 多态性 …… 163
7.4 Java 的异常处理 …… 165
7.4.1 异常与异常类 …… 165
7.4.2 异常的抛出 …… 165
7.4.3 捕获异常 …… 166
7.5 Java 的多线程 …… 168
7.5.1 多线程的定义 …… 169
7.5.2 线程优先级 …… 171
7.5.3 线程同步 …… 171
小结 …… 174
练习 7 …… 174

第 8 章 JSP 简介 …… 176
8.1 了解 JSP …… 176
8.1.1 JSP 的工作原理 …… 176
8.1.2 JSP 的特点 …… 176
8.2 Tomcat 服务器的安装和配置 …… 177
8.2.1 Tomcat 服务器的安装 …… 177
8.2.2 Tomcat 服务器的配置和测试 …… 179
8.3 JSP 的基本语法 …… 180
8.3.1 一个简单的 JSP 页面 …… 180
8.3.2 JSP 的变量、方法与表达式 …… 181
8.3.3 JSP 注释元素 …… 182
8.3.4 JSP 指令元素 …… 184
8.3.5 JSP 动作元素 …… 187
8.3.6 JSP 脚本元素 …… 197
小结 …… 198
练习 8 …… 198

第 9 章　JSP 的内置对象 …… 200
9.1　内置对象概述 …… 200
9.2　out 对象 …… 200
9.3　request 对象 …… 202
9.4　response 对象 …… 206
9.4.1　response 对象的概述 …… 206
9.4.2　response 对象的应用实例 …… 207
9.5　session 对象 …… 210
9.5.1　session 对象的概述 …… 210
9.5.2　session 对象的应用实例 …… 211
9.6　application 对象 …… 215
9.7　config 对象 …… 217
9.7.1　config 对象的概述 …… 217
9.7.2　config 对象的应用实例 …… 217
9.8　exception 对象 …… 221
9.8.1　exception 对象的概述 …… 221
9.8.2　exception 对象的应用实例 …… 221
9.9　page 对象 …… 224
9.10　pageContext 对象 …… 224
小结 …… 226
习题 9 …… 226

第 10 章　JSP 的文件操作 …… 228
10.1　File 类 …… 228
10.1.1　获取文件属性 …… 228
10.1.2　创建目录 …… 231
10.1.3　删除文件和目录 …… 232
10.2　JSP 的输入流和输出流 …… 232
10.2.1　字节流 …… 233
10.2.2　字符流 …… 235
10.3　文件的操作 …… 237
10.3.1　读取文件 …… 237
10.3.2　写入文件 …… 239
10.3.3　追加操作 …… 242
10.3.4　使用 RandomAccessFile 类 …… 242
10.4　文件上传 …… 244
小结 …… 248
练习 10 …… 249

第 11 章　JSP 访问 Web 数据库 …… 251
11.1　JDBC 简介 …… 251
11.1.1　JDBC 基本概念 …… 251
11.1.2　数据库的连接方式 …… 252
11.1.3　JDBC 常用接口 …… 255
11.2　数据库的访问 …… 260
11.2.1　插入记录 …… 261
11.2.2　查询记录 …… 263
11.2.3　更新记录 …… 265
11.2.4　删除记录 …… 266
11.2.5　JSP 访问数据库的应用实例 …… 267
11.3　数据库访问常用技巧 …… 280
11.3.1　中文字符乱码问题的解决 …… 280
11.3.2　分页显示的方法 …… 283
11.3.3　连接池的使用 …… 287
小结 …… 289
练习 11 …… 289

第 12 章　JSP 的 JavaBean 编程 …… 291
12.1　JavaBean 概述 …… 291
12.1.1　JavaBean 的简单应用 …… 291
12.1.2　访问 JavaBean 的基本语法 …… 293
12.2　JSP 页面使用 JavaBean …… 299
12.2.1　JavaBean 的属性 …… 300
12.2.2　JavaBean 的作用域 …… 302
12.3　利用 JavaBean 访问数据库 …… 307
12.3.1　JavaBean 连接数据库 …… 309
12.3.2　JavaBean 实现数据库操作 …… 311
12.3.3　访问数据的应用实例 …… 313
小结 …… 318
练习 12 …… 318

第 13 章　JSP 的 Servlet 编程 …… 320
13.1　Servlet 技术 …… 320
13.1.1　Servlet 的框架 …… 321
13.1.2　Servlet 的生命周期 …… 322
13.1.3　Servlet 的开发与部署 …… 323
13.2　JSP 的开发模式 …… 327
13.2.1　JSP Model I:JSP+JavaBean …… 328

13.2.2 JSP Model II:JSP+JavaBean+Servlet …… 328
13.3 JSP+Servlet 的应用 …… 329
13.3.1 Servlet 实现会话管理 …… 329
13.3.2 Servlet 实现文件操作 …… 334
13.3.3 Servlet 实现数据库的访问 …… 337
小结 …… 345
练习 13 …… 346

第 14 章 JSP 和 XML …… 347
14.1 JSP 生成 XML …… 347
14.1.1 JSP 直接生成 XML 文件 …… 347
14.1.2 结合 JavaBean 生成 XML 文件 …… 349
14.2 JSP 解析 XML …… 350
14.2.1 JAXP API 概述 …… 351
14.2.2 JSP 应用 DOM …… 352
14.2.3 JSP 应用 SAX …… 359
14.3 JSP 应用 XML …… 363
14.3.1 JavaBean 封装 XML 数据 …… 363
14.3.2 用户自定义标签封装 XML …… 367
14.3.3 JSP 转换 XML 文件 …… 374
14.4 JSP+XML 的应用实例:开发技术论坛 …… 378
14.4.1 技术论坛简介 …… 378
14.4.2 用户登录 …… 378
14.4.3 用户注册 …… 382
14.4.4 论坛导航 …… 385
14.4.5 用户发表新信息 …… 389
14.4.6 用户发表回复 …… 394
14.4.7 管理员的论坛管理 …… 398
小结 …… 406
习题 14 …… 406

附录 A Eclipse 与 Tomcat 的整合及使用 …… 408
A.1 Eclipse 和 MyEclipse 的安装 …… 408
A.2 Eclipse 与 Tomcat 的整合 …… 411
A.3 Eclipse 开发一个 Web 应用 …… 412

第 1 章　Web 技术基础

1.1　Internet 概述

Internet 连接了世界不同国家、地区的不同计算机，任何接入 Internet 的计算机都可以访问位于 Internet 上的共享数据资源。Internet 已经成为人们生活的重要部分。

1.1.1　Internet 的发展

Internet 是覆盖全球的信息基础设施，用户可以利用 Internet 不受地域限制的实现电子邮件的发送和接收，信息传输和查询，语言通信和图像、音频、视频的播放等活动。

ARPANET 是 Internet 的前身，是美国国防部实施的 ARPANET 计划，目的是建立一个分布式系统。为了达到这样的目的，设计了一个协议 IMPs 实现计算机发送及接收信息和数据。1968 年开始进行 IMPs 实验，并于 1969 年成功地实现了加州大学洛杉矶分校和斯坦福大学的互连，标志着网络的雏形开始形成。

1972 年第一届计算机通信国际会议上，ARPA 科学家们演示了 ARPANET，连接了 40 个不同地方的计算机。国际会议组建了 IWG（Internet 工作组），合作研究计算机互联的通信协议。1974 年，ARPA 科学家与斯坦福大学合作，开发 TCP/IP 协议。TCP/IP 协议是网络通信的核心技术，它的问世推动了 Internet 的进一步发展。与此同时，其他的通信协议与相关技术相继问世。1974 年斯坦福大学开发了 Telnet。1976 年 AT&T 贝尔实验室开发 Unit-to-Unix 协议。1979 年 Usenet 的建立，实现了 E-mail 通信与新闻组。1981 年美国纽约城市大学开发 Bitnet，实现了电子邮件列表。与此同时，美国国家科学基金会（NSF）开始着手建立提供给各大学、企业、政府机构使用的计算机科学网（CSNet）。这些技术与协议相继成为了 Internet 的基础。1982 年，互连的不同网络最终采用了 TCP/IP 作为通信标准。至此 Internet 正式形成。

Internet 发展的脚步在继续。1984 年，DNS（域名服务）出现，使得每一个主机都有一个名字，方便人们使用 Internet 访问不同地方的资源。同时，政府开始鼓励使用 Internet。英国政府于 1984 年宣布建立 JANET 网络。次年，美国国家科学基金会建立 NSFNet 网络。这些发展使得 Internet 不再仅限于计算机专业人员使用，因而使用 Internet 的人数每年翻倍增长。

1989 年 Tim Berners-Lee 提出了 WWW 概念，就是通过 HTTP（超文本传输协议）检索网站，并于 1990 年开发浏览器，使得 Internet 开始进入 WWW 时代。WWW 的推出极大地扩展了 Internet 的应用领域。20 世纪 90 年代初期，Internet 形成了事实上的网际网，各个子网分别负责自己的架设和运作，并通过 NSFNET 互连起来。直到今天，Interent 还在不断地发展。

1.1.2 Internet 的工作原理

为了保证 Internet 中不同操作系统、不同网络的计算机能够畅通无阻地交换信息，因此在 Internet 上需要有统一的协议来规范通信过程。TCP/IP 协议就是 Internet 上所采用的标准通信协议。它实际上是一组协议的总称，包含了 TCP（传输控制协议）、IP（网络协议）、FTP（文件传输协议）、SMTP（简单邮件传输协议）、ARP（地址解析协议）等众多协议内容。其中 TCP 和 IP 是保证数据完整传输的两个最重要的协议，它们在数据传输过程中首先由 TCP 协议把数据分成若干数据包，然后由 IP 协议给每个数据包标注发送主机和接收主机的地址，这样数据包就可以在 Internet 网络上进行传输了。

1. IP 地址

由于接入 Internet 的计算机数量巨大，为了使这些电脑主机能够互相识别以更好地相互通信，Internet 为每一台主机都分配了一个唯一的地址，即 IP 地址。IP 地址是识别 Internet 中的主机及网络设备的唯一标识。每个 IP 地址长度为 32 位（4B），由 4 个十进制数通过“.”分隔组成，每个十进制数的取值范围为 0～255，描述形式如：192.168.0.1。每个 IP 地址可以分为网络地址和主机地址两部分。

IP 地址可以分为五类：A 类地址、B 类地址、C 类地址、D 类地址和 E 类地址。最常用的是 A 类地址、B 类地址和 C 类地址。

（1）A 类 IP 地址。一个 A 类 IP 地址由 1B 网络地址和 3B 主机地址组成，网络地址的最高位必须是“0”，地址范围为 1.0.0.1～126.255.255.254。

（2）B 类 IP 地址。一个 B 类 IP 地址由 2B 网络地址和 2B 主机地址组成，网络地址的最高位必须是“10”，地址范围为 128.0.0.1～191.255.255.254。

（3）C 类 IP 地址。一个 C 类 IP 地址由 3B 网络地址和 1B 主机地址组成，网络地址的最高位必须是“110”。范围为 192.0.0.1～223.255.255.254。

（4）D 类 IP 地址。第一个字节以“1110”开始，它是一个专门保留的地址。它并不指向特定的网络，目前这一类地址被用在多点广播（Multicast）中。多点广播地址用来一次寻址一组计算机，它标识共享同一协议的一组计算机。

（5）E 类 IP 地址。以“11110”开始，为将来使用保留。全零（“0.0.0.0”）地址对应于当前主机；全“1”的 IP 地址（“255.255.255.255”）是当前子网的广播地址。

2. 域名

要想访问 Internet 上的信息资源必须找到存放该信息的主机，而 IP 地址是识别主机的唯一标识，因而也就必须记忆相应的 IP 地址。但是对大多数人来说记忆众多主机的 IP 地址并不是一件容易的事情，所以 TCP/IP 协议提供了域名管理系统 DNS（Domain Name System），它为每个主机分配字符名称，也就是域名，访问网络时该系统会自动实现域名与 IP 地址的转换。当要访问清华大学网站时，只需要在地址栏输入 www.tsinghua.edu.cn 就可以了。

Internet 中域名采用分级命名的机制，基本结构如下：

主机名.三级域名.二级域名.顶级域名

因此可以分析域名 www.tsinghua.edu.cn，其主机名为“www”，三级域名为“tsinghua”表示

"清华大学",二级域名"edu"表示"教育机构",顶级域名"cn"表示"中国"。

3. URL

统一资源定位符 URL(Uniform Resource Locator)是对可以从 Internet 网上得到的资源的位置和访问方法的一种简捷的表示。URL 提供了待访问信息资源在 Internet 上的准确位置,描述了要访问该资源所使用的访问方式、提供 Web 服务的主机名、待访问文档在主机上的路径和文件名,以及客户机与远程主机通信时使用的端口号。它的基本格式如下:

<访问方式>://<主机名>:<端口号>/<文件路径>

其中常用的访问方式有 http(超文本传输协议 HTTP)、ftp(文件传输协议 FTP)等。对于某些资源在访问时需要给出服务器提供的端口号,但是在一般情况下服务器都采用标准保留端口号,如 HTTP 的保留端口号是 80。对于 URL 的描述＜访问方式＞和＜主机名＞是必需的,＜端口号＞和＜文件路径＞有时可以省略。如当我们要通过浏览器访问清华大学出版社的主页时可以在浏览器的地址栏输入以下 URL 地址:http://tup.tsinghua.edu.cn。

1.1.3 Internet 的应用领域

Internet 现在已经成为人们获取和传播各种信息的一个重要途径,它已经深入到日常生活的各个方面。随着 Internet 技术的不断发展,它所提供的服务范围和应用领域也逐步扩大。Internet 提供的基本服务包括以下几方面。

1. WWW 服务

WWW 服务是 Internet 上使用最广泛最方便的服务类型。它主要由网页组成,通过超链接将不同页面联系在一起,集合了海量的信息资源。WWW 提供了遍布全球的信息检索,功能十分强大,能在线展示文字、图像、声音及动画等种类繁多的超媒体文件,用户可以享受在线欣赏影片、音乐、网上购物等服务。

2. E-mail 服务

E-mail(电子邮件)服务是 Internet 上使用广泛的一种服务。它可以发送文本文件、图片、程序等信息。只要有电子邮件地址,通过它用户就可以和身处任何地方的朋友联系并传递信息,它是一种快速、简单、廉价的通信方式。

3. FTP 服务

文件传输服务 FTP(File Transfer Protocol)也是 Internet 提供的基本服务之一,它允许 Internet 上的用户将文件在不同计算机之间进行传送。FTP 采用客户机/服务器的工作模式,可以在 Internet 上传输二进制文件、文本文件、图像声音文件等几乎任何类型的文件。用户使用 FTP 服务可以通过 DOS 命令行方式、资源管理器方式和 FTP 工具进行访问。为了增强文件的上传和下载功能,现在已经开发出很多 FTP 工具软件,较为常见的有 CuteFTP、LeapFTP 等。

4. BBS 服务

Internet 提供的电子公告牌系统 BBS(Bulletin Board System)服务使 Internet 用户能够在网上方便地发布信息,同时可以展示自己对事物的观点和看法。很多 BBS 站点都提供 WWW 和 Telnet 两种访问方式。

5. USENET 服务

USENET 是一个用于发布各种新闻和文章供用户阅读、讨论和发表评论的电子公告牌，是一个针对某个主题的网上新闻组，主题的内容可以涉及文艺、体育、政治、哲学等各个方面。用户可以选择订阅自己感兴趣的新闻，也可以参加新闻组的讨论发表自己的看法。

6. E-Commerce 服务

电子商务 E-Commerce 也是现在日趋成熟同时也是很受 Internet 用户欢迎的一种服务。它是指通过 Internet 进行的产品或服务的销售行为，是一种基于 Web 的商务活动。它包含 B2B、B2C、B2G 等多种形式。由于电子商务交易方便、价格通常较低廉、商品信息获取快速直接，因此越来越受到 Internet 用户的喜爱。

1.2 Web 技术概述

Web 全称为 World Wide Web(简称 WWW，也就是万维网)，是 Internet 提供的一种信息服务。如今，Web 已经深入到人们生活的各个方面，几乎所有的信息技术领域都或多或少受到 Web 的影响，人们也逐渐认识到 Web 技术的重要性。

1.2.1 Web 技术简介

Web 汇集了各种不同类型的信息，它的页面颜色丰富、包含文字、图形、动画、声音和视频等多种信息内容，因此在 Web 开发时也需要使用到对不同类型信息的处理技术。Web 站点的开发可以分成客户端和服务器端两部分，客户端主要用于显示信息内容，也就是我们浏览的 Web 页面，服务器端程序的主要功能是对所需信息进行处理。

1. Web 客户端开发技术

常用的 Web 客户端开发技术有 HTML、XML、CSS 和脚本语言等。

(1) HTML。对于 Web 页面的设计开发，用到最多的是 HTML(HyperText Markup Language，超文本标记语言)，它是整个 Web 技术的基础。HTML 语言是一种标记语言，它通过标记来描述页面上的文字、影像、图片等内容。我们可以使用任何文字编辑软件来编辑 HTML 代码，常用的开发软件如 Dreamweaver 等都是制作 HTML 页面的不错选择。

(2) XML。由于 HTML 只能使用其已定义的各种标记，难以展开，交互性和语义性较差，因此人们在 HTML 基础上加以改进形成了 XML(eXtensible Markup Language，可扩展标记语言)。XML 可以自定义标记，描述的是文档的结构和意义，它本身不描述文档的显示方式。所以要想使 XML 文档在浏览器中按特定样式显示，必须要有相应的样式文件，如 CSS 文件或 XSL 文件等。值得注意的是，XML 也是一种服务器端的技术，多用于服务器端信息配置、数据交换、内容管理、Web 应用等各方面。

(3) CSS。CSS(Cascading Style Sheets，层叠样式表)的主要工作就是描述 Web 页面的显示风格和样式，使用它可以使得 Web 页面的显示内容与显示样式分开，有利于对页面文件的维护。同时 CSS 还提供了丰富的页面显示样式，使得页面的视觉效果更加丰富多彩，提高了网站的吸引力。

(4) 脚本语言。常用的脚本语言有 VBScript 和 JavaScript，它们都是由浏览器解释执行的脚本语言，可以处理和使用 HTML 页面中的各种对象。其中 VBScript 具有与 Visual Basic 几乎相同的语法结构，而 JavaScript 和 Java 语言的语法结构类似，但比 Java 更简单有

效。它们不仅可以作为客户端程序的开发技术，也可以应用于服务器端程序的开发。在客户端使用脚本语言必须嵌入到 HTML 文档中，随页面一起被下载到客户端并由浏览器解释执行。使用脚本语言的目的是为了实现在 Web 页面与用户的交互功能，产生动态效果。

2. Web 服务器端开发技术

目前常用的 Web 服务器端开发技术有 JSP、ASP、PHP 和 ASP. NET 等。

(1) JSP。JSP(Java Server Pages)是由 Sun MicroSystem 公司于 1996 年推出的一种动态网页技术标准。JSP 页面由 HTML 代码和嵌入其中的 JSP 代码组成，是基于 Java Servlet 及整个 Java 体系的 Web 开发技术。JSP 页面被客户端请求后由服务器对该 JSP 代码进行处理，然后将运行结果返回给客户端的浏览器。JSP 具备了跨平台、通用性好、安全可靠等特点，是当前较为流行的一种服务器端 Web 程序开发技术。

(2) ASP。ASP(Active Server Pages)是由 MicroSoft 公司在 1996 年年底推出的一种运行于服务器端的 Web 应用程序开发技术。ASP 没有提供专门的编程语言，而是允许用户使用已有的脚本语言(如 VBScript 或 JavaScript)来编写 ASP 的应用程序。通过 ASP 我们可以将 HTML、ASP 指令和 ActiveX 组件结合以建立动态交互而且高效的 Web 服务器应用程序。

(3) PHP。PHP(Personal HomePage tools)由 Rasmus Lerdorf 于 1994 年提出，1995 年发布了第一个版本。PHP 语法借鉴了 C、Java 和 Perl 等语言，它可以嵌入到 HTML 中，更好地对页面进行控制。

(4) ASP. NET。2004 年 MicroSoft 公司提出了. NET 架构，ASP. NET 就是其中一部分，它提供了在分布环境下进行 Web 应用开发的环境和工具。ASP. NET 与 ASP 存在很大的差别，它的运行效率较高，并且支持多种开发语言，如 Visual Basic. NET、C#. NET、Visual C++. NET 等。

1.2.2 Web 的工作模式

Web 是 Internet 提供的一种服务，是基于 Internet、采用 Internet 协议的一种体系结构。只要有一台计算机与 Internet 相连，就可以通过它访问处于 Internet 上任何位置的 Web 站点。Web 的内容保存在 Web 站点(服务器)中，用户可以通过浏览器访问 Web 站点，获取自己所需的各种信息，这些信息都是彼此关联的文档，通过超链接将不同的页面信息连接在一起。Web 所存放的信息是超文本类型的，包含了文字、图形、音频视频等多种内容。

超文本传输协议 HTTP(HyperText Transfer Protocol)是专为 Web 设计的网络协议，它是位于 TCP/IP 参考模型的顶层—应用层的协议，也是 Internet 上能够可靠地交换文件信息的重要基础。HTTP 是用于从 WWW 服务器传送文件到本地客户端浏览器的传送协议。从 TCP/IP 参考模型的层次角度我们也可以将它理解为面向事物的应用层协议。HTTP 规定在客户机和服务器之间的每次交互都是由一个 ASCII 码串构成的请求和一个类 MIME 的响应组成的。

HTTP 协议是基于请求/响应的工作模式，当 Web 浏览器和服务器间用 HTTP 协议来传送文档时，它的工作过程可以理解如下(其工作过程如图 1-1 所示)。

(1) 用户启动客户端浏览器，在浏览器中输入要访问的 URL 地址，由浏览器向 DNS 请

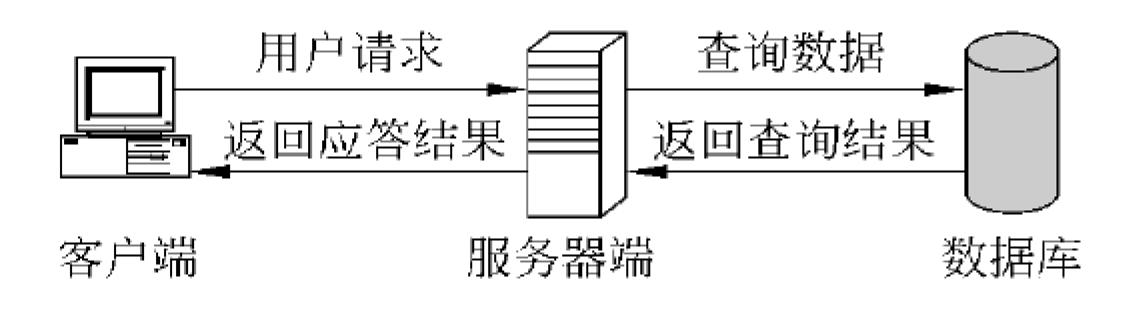


图 1-1　Web 工作示意图

求解析该 URL 对应的 IP 地址,并向该 IP 地址对应的服务器发送建立连接的请求。

（2）浏览器与服务器建立 TCP 连接。

（3）服务器给出响应,将被访问文件发回给浏览器;具体的执行过程有如下几种情况。

① 服务器根据客户端发送的请求找到相应文件,如文件是 HTML 文档,则将该文档直接返回给客户端。

② 如果文件中包含 JSP、ASP 或 PHP 程序,则由 Web 服务器运行该程序并把运行结果返回给客户端。

③ 如果程序中包含对数据库的操作,则服务器将指令发送给数据库驱动程序,由数据库驱动程序执行相关指令并将执行结果返回给 Web 服务器,然后再通过服务器将数据运行结果嵌入页面并将完整的 HTML 页面返回给客户端浏览器。

（4）TCP 连接被释放。

（5）客户端和服务器断开连接。

小　　结

本章介绍了 Web 编程的基础知识,包括 Internet 的基本知识和它所提供的基本服务;IP 地址、域名、URL 的基本概念;Web 开发的基本技术、Web 的基本工作原理;HTTP 协议的基本概念和工作原理。通过本章的介绍,希望大家对 Web 的相关技术以及 Web 的工作过程有一个初步的了解,能够更好地进行下面内容的学习。

练　习　1

1. 填空题

（1）目前常用的 Web 服务器端开发技术有________、________、PHP 和 ASP.NET 等。

（2）Web 采用________的工作模式。

（3）________是识别 Internet 中的主机及网络设备的唯一标识,它的长度为________。

（4）TCP/IP 协议提供了________,它为每个主机分配字符名称,访问网络时该系统会自动实现域名与 IP 地址的转换。

（5）HTTP 中文全称为________,是基于________的工作模式。

2. 选择题

（1）使用下面技术编写的代码可以通过浏览器直接执行的是________。

A. ASP　　B. HTML　　C. JSP　　D. PHP

（2）下面对 Web 的描述错误的是________。

A. Web 全称为 World Wide Web。

B. Web 是 Internet 提供的一种服务。

C. Web 是基于 Internet、采用 Internet 协议的一种体系结构。

D. Web 所存放的信息是文本类型的。

3. 简答题

(1) 要开发一个 Web 应用程序,常用的开发技术有哪些?

(2) 什么是 URL,它的基本描述格式是什么?

(3) 简述 HTTP 的工作原理。

第 2 章　HTML 和 XHTML

2.1　超文本标记语言 HTML

HTML 是 Hypertext Markup Language 的缩写，表示超文本标记语言，是万维网的基本描述语言。其中，Hypertext 表示"超文本"，能使文件展示包含文本和图像在内的多种媒体形式。同时，"超文本"意味着突破了传统访问文件必须依从的线性访问方式，可以在多个 HTML 文件之间跳跃访问，不必依从特定的顺序。其次，单词"Markup"代表标签，实际上 HTML 是一个标签系统，它定义了各种关于内容展示的标签，如标签可以将包含的文本内容以粗体文本格式显示。通过这些格式提高了文本显示的可用性。最后，HTML 是一种语言，利用这种语言可以定义格式形式截然不同的 HTML 文件。通常会把 HTML 文件或具有这种内容格式的文件称为网页。

HTML 文件定义网页的内容以及显示格式，具体的实现是依赖于网络浏览器如 IExplorer、Netscape 对网页文件的解释。通过网络浏览器展示文件的文本和图像等多种形式的媒体信息。尽管 HTML 文件表达的内容非常丰富，但是书写形式非常简单。开发人员可以使用任何一个文本编辑器甚至最简单的记事本都可以实现。目前常用的网页开发工具有 EditPlus V 3.0、DreamWeaver CS3、Microsoft Office SharePoint Designer 2007 等。

2.1.1　HTML 的发展历史

要学习和掌握 HTML，就不得不先了解 HTML 的发展历史。HTML 产生于欧洲核能研究所。在欧洲核能研究所的不同研究项目小组之间需要信息共享。由于早期的网络中文件的共享只能通过访问特定计算机下载文件来实现的。1989 年，工作在该所的计算机服务组的 Tim Berners-Lee 希望改变这一种不方便的访问状况，提出不同地方不同平台的计算机用户可以通过链接而不是直接下载文件来实现文件的共享。为此，1990 年他开始着手研究 WWW 项目，将设计的初级浏览和编辑系统在网上合二为一，他创建了一种快速小型超文本语言来为项目服务。最初的 HTML 是基于标准通用标签语言 SGML（Standard Generalized Markup Language），具有平台独立、超文本和精确化文档结构。

但是早期的 HTML 以文本格式为基础的简易的语言，所定义的有限标签已经限制了 Web 的一些应用。伴随着互联网的迅速发展，人们发现简单的文本标签并不能表现更丰富的内容。在 1993 年，University of Illinois 一位学生 Marc Andreessen 研发著名的 Mosaic 浏览器，并在浏览器中加入了<img>标签支持图像的显示。以此为转折，一些新的标签不断推出，也同时刺激着互联网的迅猛发展。各个浏览器的研发小组制定各自的 HTML 标准。但是，这些新标准带来一些问题：一些新标签的出现，并没有得到长久的支持和应用，而且一些新标签也没有很好地符合 SGML 的标准。导致用户通过支持不同标签的不同浏览器访问同一个文件，产生不同的显示效果。在这种混乱的情况下，出现要求统一 HTML

标准的声音。

1994 年，Tim Berners-Lee 创建了万维网联盟（World Wide Web Consortium，W3C）。W3C 组织致力于 Web 技术的标准化。1995 年欧洲核能研究所主持下在瑞士日内瓦举行的第一次 WWW 会议上成立了一个 HTML 工作小组。它的主要任务是把 HTML 形式化成为一种 SGML DTD，称之为 HTML 2.0 标准。

但是，各个方面对该标准看法并不一致。特别是在一些计算机企业在浏览器市场领域的竞争，对 HTML 的发展造成了一些比较混乱的现象。最明显的是由于市场行业标准不一致，导致用户访问同一网页得到的渲染形式不同。W3C 联盟在 1996 年对 WWW 技术中出现的一些成果进行总结，并领导制定了著名的 HTML 3.2 标准。1997 年 HTML 3.2 标准正式说明问世。HTML 3.2 标准和 HTML 2.0 标准完全兼容，并添加了当时对多媒体技术的支持的标签，如 applet 标签、表格标签等。HTML 3.2 最重要的特点是跨工业。这意味着不同计算机浏览器开发企业都遵循该标准。HTML 技术在内容渲染方面功能强大，但这同时使 HTML 文件变得难以理解。为了适应互联网发展趋势，业内提出文件数据的定义和内容的表现方式分开。HTML 4.0 开始步入 Web 世界。

HTML 4.0 及之后的版本是继承了已有的 HTML 版本的特点，并伴随技术的发展出现一些新的特性。

（1）将文档结构和文档展示格式进行区分。并主张利用层叠样式表 CSS（Cascading StyleSheets）实现文档展示格式。

（2）增加对表单的控制能力，提高表单的性能。

（3）增加表格的属性，提高表格的应用能力。

（4）允许文档中支持对象。

（5）增加图像的映像功能等特点。

由于 HTML 是脱胎于 SGML 语言的，继承了 SGML 的强大功能和灵活性，但同时也带来一个问题：由于 SGML 对适应它的语法分析程序要求较高，HTML 也继承了这个问题。XML 可扩展标记语言这一个全新的元语言开始进入人们的视野。XML 力求克服 SGML 的语法分析的复杂性，又希望具有 SGML 的灵活和强大功能。与 HTML 相比，XML 表达数据的灵活方便，但是与人们长期应用 HTML 的习惯产生严重冲突。1998 年许多计算机组织和企业开始探讨重新定义 HTML，以符合未来对 XML 应用的要求。在 2000 年，W3C 联盟推荐了 XHTML 1.0。XHTML 是用 XML 语法优化和改进的 HTML。除了对 XML 解释方面不同，实际上 XHTML 1.0 和 HTML 4.01 没有太大的区别。但是 XHTML 和 HTML 至此开始了各自的独立发展。

2007 年，WHATWGa（Web Hypertext Application Technology Working Group，网络超文本应用工作组）推出了 HTML 5.0 标准草案。该标准增加了新的特性，主要帮助 Web 上日益增多的网络著述、表达等方面的应用。特别是对一些网络博客来说，提高了编辑文档内容的便利性。HTML 5.0 标准的问世，进一步提高了 Web 的应用。

2.1.2 HTML 文件的页面结构

在本章中除了特别说明，将使用 HTML 4.01 标准。一个 HTML 文件可以保存扩展名为 html 或者 htm。为了了解 html 文档的基本页面结构，先来看一个简单的 html 页面，见

程序清单 2-1 的 firstPage. html 文件，该文件的执行结果见图 2-1。

程序清单 2-1：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<!--firstPage.html 源文件代码-->
<html>
<head>
    <title>第一个网页标题</title>
</head>
<body>
    <p align="center"><b>显示第一个网页!</b>
</body>
</html>
```



图 2-1　第一个 html 的例子

从程序清单 2-1 定义的 firstPage. html 文件可以看出，一个合法有效的 HTML 有两大部分：文档类型说明和 HTML 的基本元素。

1. 文档类型说明

一个合法的 html 文件必须说明使用的 html 的版本。html 版本是在 DOCTYPE 元素中声明特定的 html 的文档类型定义 DTD(Document Type Definition)实现的。DTD 定义了一系列的规则集，它的作用在于浏览器对网页执行的显示方式的不同。设置正确的 DTD 文档类型会使浏览器展示网页更加迅速合理。文档类型说明形式包含了两部分内容，格式如下：

```
<!DOCTYPE HTML PUBLIC "说明文档类型部分" "系统标识部分">
```

其中，各字段含义如下：

(1) 说明文档类型部分：说明文档类型 DTD；

(2) 系统标识部分：确定浏览器寻找 DTD 的统一资源定位 URL。

HTML 4.01 定义了 3 种类型的 DTD。

① HTML 4.01 Strict DTD(严格类型)。排除了淘汰了过时的所有元素和属性。用<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">说明。

② HTML 4.01 Transitional DTD(过渡类型)。具有除 FrameSet 外的全部内容，包含

了 HTML 4.01 Strict 的标签和属性。用＜!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"＞说明。

③ HTML 4.01 Frameset DTD(框架类型)。包含了 HTML 4.01 Transitional DTD 的所有标签和属性,以及可以使用框架。用＜!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd"＞说明。

注意:

(1) ＜!DOCTYPE＞并不是 html 基本结构的组成。

(2) 有些 html 文件忽略了 DTD 的说明,并不影响网页内容的显示,但是会影响网页执行效率。

(3) ＜!DOCTYPE＞必须是第一个非注释内容的,不能在其他标签之后。

2. html 文件的基本结构

标签(也称标记)是 HTML 网页的基础,一个 HTML 文档往往包含了许多标签,这些标签由尖括号"＜"和"＞"包含具体含义标识符构成,标识符不区分大小写。HTML 标签一般成对出现,有开始标签也有结束标签,结束标签比开始标签前置"/"符,形如"＜/标记＞"。具体的标签说明了文档定义结构,而开始标签和结束标签之间的是表达文档内容的文本信息。在运行时,浏览器对标签进行解释执行,对于能正确识别的标签,会将标签之间的文档内容正确表达。如果浏览器对于不能识别的标签,则会忽略它们。标签可以组成 html 文件的元素。

html 的元素是一种包含文本或嵌套其他对象实体的对象实体。实际上,一对合法标签以及它们之间的文本就是一个元素。html 文件的元素是一种特殊的对象,它可以嵌套其他对象在内。在程序清单 2-1 所示的 p 元素,p 元素又嵌套了 b 元素。b 元素包含了一段文字信息。

元素作为一个对象实体,也具有自己的属性。属性就是包含在标签中信息内容。通过属性,可以突出标签的某些特定的性质,或修改元素的已有属性。在 firstPage.html 中,定义 p 元素的排列属性 align 的值为"center",表示该元素包含的文字内容居中显示。

html 文档有许多元素构成,但是构成 html 文档基本结构的主要三种元素是:html 元素、head 元素和 body 元素,如图 2-2 所示。

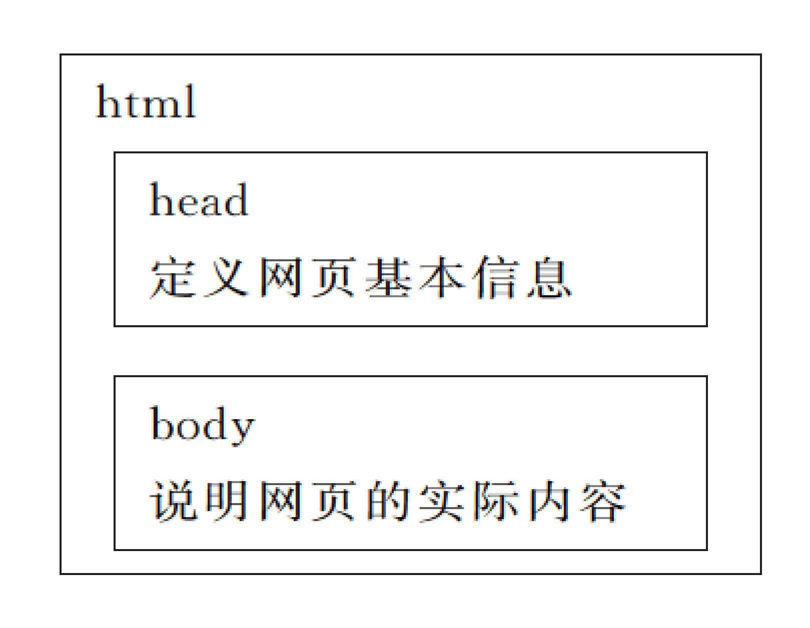


图 2-2 html 文件的基本结构

html 元素将 head 元素和 body 元素包围起来。html 元素实际上是定义了整个网页。网页的所有内容都包含在 html 元素中。head 元素定义了 html 文件的首部节,用以说明关于网页的基本信息,以及网页所使用的脚本语言、著作人等内容,例如利用 title 元素定义网页的标题。通常 head 元素定义的内容不会在网页本身中显示,它所侧重的只是对网页本身性质的描述和说明。body 元素是正文节,在这里放置了网页的具体内容。

程序清单 2-1 的 firstPage.html 展示了一个简易网页。该网页定义为 HTML 4.01 Strict 的文档类型。它的 head 元素嵌套定义标题标签 title,通过 title 声明了一个网页标题,运行该网页,浏览器的标题栏的确是出现了"第一个网页标题"的文字信息。在同一个网页的正文节中说明了"＜p＞＜b＞显示第一个网页! ＜/b＞＜/p＞"。p 元素嵌套了一个 b

元素，网页中的主要信息则以粗体形式显示文本信息“显示第一个网页!”，如图 2-1 所示。

在这个例子中，还有一类元素是 html 的注释。HTML 注释是包含＜!--和--＞之间的内容。注释的内容不会在网页中显示，一般是为开发网页提供帮助和提示的作用。

2.1.3 HTML 的基本标签

HTML 的基本标签是构成一个网页的基本内容，HTML 的基本标签除了在上一节中已经介绍的定义 html 网页结构的 html 标签、head 标签和 body 标签之外，还包括嵌套在这些标签中的标题标签、段落标签以及换行标签。HTML 的基本标签见表 2-1。这些标签是组成一个简易网页常常出现的重要内容。在本节中，将对段落标签、头标题标签、换行标签以及水平线标签做一个详细介绍。

表 2-1 HTML 的基本标签

基本标签	说　　明	基本标签	说　　明
html	定义一个 html 页面	h1～h6	定义各种规格的头标题
head	定义 html 的首部实体	br	换行
title	定义一个网页的标题	hr	定义一个水平线
body	定义正文实体	p	定义段落
div	定义组合块	span	定义文档内的内联元素

1. 头标题标签

头标题是出现在 body 元素中常常定义一段文字的标题。html 提供了 6 种头标题，从最大号的文字 h1 到最小号的文字 h6。它们往往起到强化网页内容的作用。下列的源程序清单 2-2 以及运行结果可以很好地说明它们的应用，运行结果如图 2-3 所示。

程序清单 2-2：

```
< !-- headings.html -->
  < !DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
  <html>
  <head>
    <title>头标题示例</title>
  </head>
  <body>
    <h1>头标题 1</h1>
    <h2>头标题 2</h2>
    <h3>头标题 3</h3>
    <h4>头标题 4</h4>
    <h5>头标题 5</h5>
    <h6>头标题 6</h6>
  </body>
  </html>
```

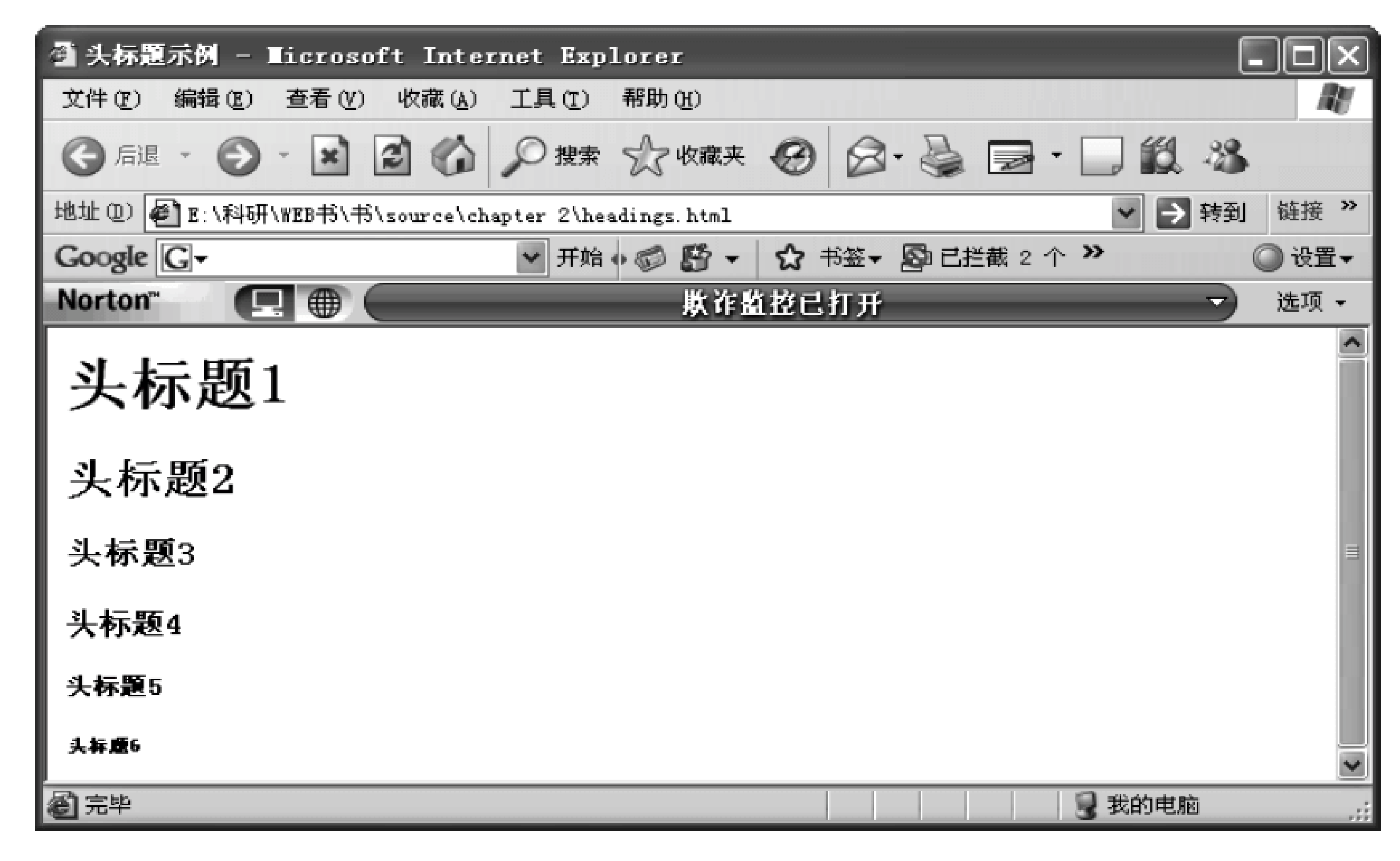


图 2-3　h1～h6 的运行示例

2. 段落标签

段落标签是用来定义 body 元素正文段落内容的标签。段落定义包含在 p 元素内部，通常作为展示网页的内容。p 标签定义的段落有一个特点，就是在此段落之前和之后会自动换行。根据 p 标签这一特性，也常用它处理文本换行。观察下列的源程序清单 2-3 以及它的运行结果，可以发现，不用 p 标签定义的文字信息在一行内显示；而定义在 p 元素内的段落文字会自动换行显示，运行结果如图 2-4 所示。

程序清单 2-3：

```
<!-- p.html-->
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
  <html>
  <head>
    <title>段落标签的示例</title>
  </head>
  <body>
    <p>p标签定义的一段文字</p>
    <p>p标签定义的另一段文字</p>
    没有 p标签定义的一段文字
    没有 p标签定义的另一段文字
  </body>
  </html>
```

3. 换行标签

p 标签可以为段落自动换行，但是在段落内需要换行则要求助于专门的换行标签＜br＞。＜br＞标签的主要作用就是在 html 文件中执行换行功能，用以结束无论是段落内还是段落外的一行内容。＜br＞标签是一种特殊的非封闭标签，它没有结束标签。如果要写成＜/br＞则是错误的表达形式。将程序清单 2-3 改写成下列的程序清单 2-4 的形式，可以看到，最后两行的文字出现了换行。br 标签的换行与 p 标签的换行不同，p 标签的换行带有一定的段落格式，段落与段落之间存在间距。而 br 标签仅仅是换行。行与行之间的文字



图 2-4　p 标签定义段落运行示例

不存在空隙，运行结果如图 2-5 所示。

程序清单 2-4：

```
<!-- pbr.html-->
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
  <html>
  <head>
    <title>段落标签的示例</title>
  </head>
  <body>
    <p>p标签定义的一段文字</p>
    <p>p标签定义的另一段文字</p>
    没有 p标签定义的一段文字<br><!--增加换行标签-->
    没有 p标签定义的另一段文字
  </body>
</html>
```



图 2-5　br 标签的运行示例

4. 水平线标签

水平线标签可以写成<hr>，它同样是一个非封闭的标签，没有结束标签。该标签的作

用是为网页增加一个水平线，由于水平线的长短粗细可以通过设置属性来调整，所以有时也把水平线称为水平尺。hr 标签常用来分割网页中主题不同的内容。请看程序清单 2-5，来了解 hr 标签的用法，运行结果如图 2-6 所示。

程序清单 2-5：

```
<!-- hr.html-->
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
  <html>
  <head>
    <title>水平线的示例</title>
  </head>
  <body>
    插入水平线一
    <hr>
    插入水平线二
  </body>
</html>
```



图 2-6　水平线运行示例

5. div 标签和 span 标签

由于从 HTML 4.01 问世以来，就强调文档内容和表现进行区分，往往用 div 和 span 标签结合 CSS 层叠样式表来表现文档。

div 标签和 span 标签都可以定义文档中的节。div 标签和 span 标签通常用来组织文档，将文档中的部分内容作为一个整体处理。div 标签和 span 标签本身没有太大的意义。但是，在与 CSS 层叠样式表结合会产生强大的作用。具体应用见第 3 章。

2.1.4　超链接

html 文件的一个重要特点，就是它能够在不同网页跳转访问。html 中定义实现这样一个功能的锚点标签 a。锚点标签可以定义一个锚点，它的作用有两方面：

(1) 锚点标签<a>和属性 href 结合，实现超链接，使得一个网页能链接到包括其他的网页、图像、音频等在内的其他网络资源。

(2) 锚点与属性 name 或 id 结合，访问本页面的一个书签。

表 2-2　a 标签的主要属性

属　性	属 性 值	说　明
href	同一资源定位 URL	链接到指定的 URL
name	指定名字	定义锚点的名称
target	_top	定义打开网页的位置，其中： _top 表示 url 指定的网页将在最初的窗口完全打开； _blank 表示 url 指定的网页将在一个新的窗口打开； _parent 表示 url 将在上级窗口中打开； _self 表示 url 指定的网页将在当前窗口打开
	_blank	
	_parent	
	_self	
type	mime 类型	确定 url 的 MIME(多用途互联网邮件扩展)类型

1. a 标签结合 href 定义一个超链接

a 结合 href 定义一个超链接的形式：

```
<a href=url 地址>链接文字描述</a>
```

当用户用鼠标单击“链接文字描述”时，网页会跳转到 url 地址指定的位置，如果 url 指定的网络资源并不存在，会显示链接失败的页面。

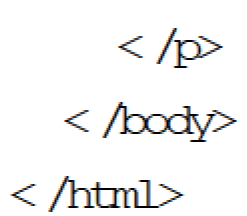
2. a 标签与属性 name 结合定义书签

在 html 4.01 标准中，a 标签可以和 name 定义一个书签。然后可以通过 a 标签结合 href 来访问已经定义的书签。但是，浏览书签名确定的位置时，需要在 URL 地址处添加一个“#”来达到访问的目的。如果一个网页的内容非常多，超过一个窗口显示的长度，就可以通过这种方式来返回到本页面的指定位置。

为了了解锚点标签的作用，请看程序清单 2-6，运行结果如图 2-7 所示。

程序清单 2-6：

```
<!-- ademo.html-->
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
  <html>
  <head>
    <title>锚点标签的示例</title>
  </head>
  <body>
    <p>
     <a name="url1">欢迎使用超链接：</a><br>
       请访问<a href="http://www.yahoo.com.cn"  target="_self">雅虎中文站点</a>!
       还有许多学校站点没有访问：<br>
            清华大学<br>北京大学<br>中国人民大学<br>
            北京交通大学<br>北京科技大学<br>北京化工大学<br>
            北京邮电大学<br>中国农业大学<br>北京林业大学<br>
            北京中医药大学<br>北京师范大学<br>北京外国语大学<br>
       <a href="#url1">返回</a><br>
       <a href="mailto:chanyee@yahoo.com.cn">和我联系</a>
```



```
    </p>
  </body>
</html>
```



图 2-7　单击返回后出现的运行结果

运行 ademo.html 网页，如果单击“雅虎中文站点”链接，会导航到雅虎中文站点处。同时，由于 target 属性为“_self”，是在一个当前窗口中打开的网页。如果单击“返回”，页面会回到打开窗口的初始状态。观察浏览器的地址栏，可以看到 url 链接添加了 # url1 的内容。如果链接的 URL 是以“mailto:邮件地址”的形式出现，这意味着单击链接，会和一个邮件地址链接，打开本机中默认的邮件处理软件。迅速进入邮件编辑状态。

2.1.5　列表

网页中包含大量的信息，可以用列表突出大量信息的不同。HTML 提供了三种列表的制作：无序列表、有序列表和定义列表。

1. 无序列表(Unordered List)

无序列表展示的是一个明细表，表中的各项目没有明确的数字或字母标志表现顺序，而是用项目符号来突出彼此的不同。由于无序列表具有这种特点，有时也称无序列表为符号列表。

无序列表是通过标签 ul 和 li 来实现定义的。其中，ul 定义一个无序列表，而 li 标签定义特定的列表项目。基本形式如下：

```
<ul>
    <li>项目 1</li>
    <li>项目 2</li>
    …
    <li>项目 n</li>
</ul>
```

标签 ul 和 li 在定义无序列表时，可以通过属性 type 来修改项目符号。属性 type 具有

三种值：disc、circle 和 square，分别表示实心圆、空心圆和实心方块三种符号。其中，ul 标签定义无序列表中所有的项目符号，而 li 标签定义指定项目的项目符号。

注意：但是，由于 html 4.01 标准提出页面结构与页面展示格式分开，关于所有格式定义已经不赞成使用，包括项目符号的定义，而是通过样式表如 CSS（见第 3 章）来实现。

标签 ul 和标签 li 组合还可以表示嵌套列表，即在一个列表内部的项目中嵌套定义一个列表，该列表为下一层的列表。通过这种方式可以交错地显示项目内容。见下列的程序清单 2-7，运行结果如图 2-8 所示。

程序清单 2-7：

```
<!--  unorderlist.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>无序列表示意</title>
</head>

<body>
列表示例：
<p>
例一：无序列表->计算机科学系的专业
  <ul>
    <li>计算机科学与技术</li>
    <li>电子商务</li>
    <li>计算机网络</li>
    <li>计算机游戏</li>
  </ul>
</p>
<p>
例二:嵌套列表->计算机科学与技术专业课程
  <ul>
    <li>2003-2004 学年</li>
        <ul>                            <!--嵌套定义一个新层的无序列表-->
          <li>计算机科学导论</li>
            <ul>                        <!--嵌套定义下一个新层的无序列表-->
              <li>实验一：熟悉 Windows XP 环境和操作</li>
              <li>实验二：Word 的应用</li>
            </ul>
          <li>高级语言程序设计</li>
          <li>汇编语言程序设计</li>
        </ul>
</ul>
</p>
</body>
</html>
```

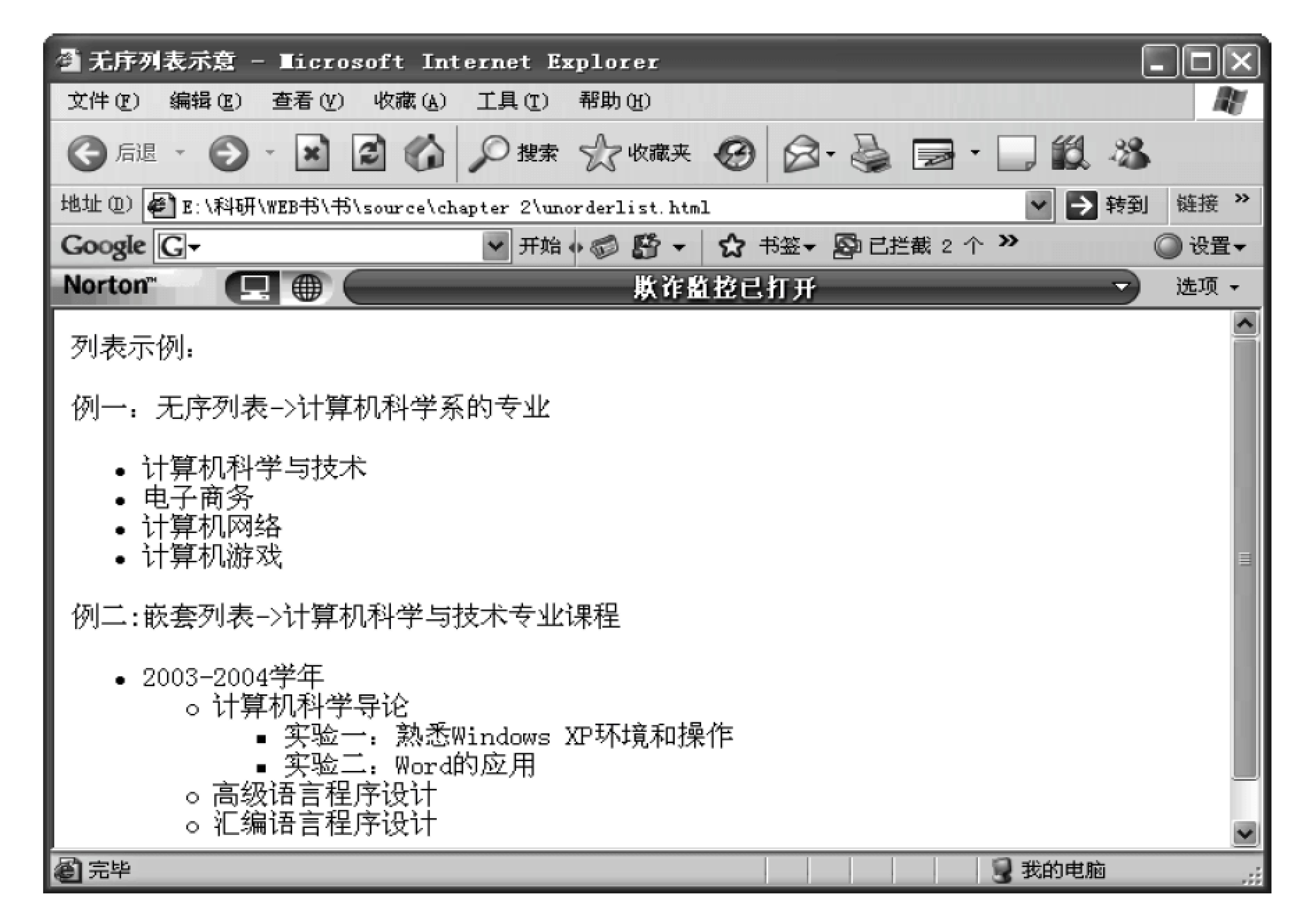


图 2-8　无序列表运行示例

2．有序列表(Ordered List)

有序列表同样也可以展示明细表,但是表中的项目可以用数字或字母突出项目出现的顺序以及所属的类别。这种显示方式以序列的方式突显了项目的类别的不同。

标签 ol 和标签 li 组合可以定义有序列表。标签 ol 的作用是声明定义一个有序列表,li 标签的作用仍是定义列表中的项目。形如：

```
<ol>
  <li>项目 1</li>
  <li>项目 2</li>
  …
  <li>项目 n</li>
</ol>
```

有序列表中是通过数字或字母的顺序体现列表中各个项目的序列的。ol 标签有两个常见的属性 type 和 start,见表 2-3,可以对项目序列进行设置。

表 2-3　ol 标签的主要属性

属　性	属 性 值	说　　明
type	A	表示列表项目的编号:A 表示编号顺序(A,B,C,…);
	a	a 表示编号顺序(a,b,c,…);
	Ⅰ	Ⅰ表示编号顺序(Ⅰ,Ⅱ,Ⅲ,…);
	ⅰ	ⅰ表示编号顺序(ⅰ,ⅱ,ⅲ,…);
	1	1 表示编号顺序(1,2,3,…)
start	开始的编号	指定编号字符的起始值

在有序列表中,标签 li 支持 type 属性的值成为 A,a,I,i 和数字 1,表示项目的编号。具

体要求同表 2-3 中 ol 标签的 type 属性。

同样，有序列表也可以定义嵌套列表，同样是在定义的一个项目嵌套定义下一层的列表。列表的形式可以是有序列表也可以是无序列表。具体应用可以参看源程序清单 2-8，运行结果如图 2-9 所示。

程序清单 2-8：

```
<!--orderlist.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>有序列表示意</title>
</head>
<body>
<p>
有序列表 1
  <ol>
    <li>计算机科学导论</li>
    <li>数据结构</li>
    <li>操作系统</li>
  </ol>
</p>
<p>
有序嵌套列表 2
 <ol>                                  <!-- 定义有序列表 -->
    <li>2003—2004 学年</li>
        <ol type= "a">                 <!-- 定义下一层有序列表 -->
          <li>计算机科学导论</li>
            <ul>                       <!-- 定义再下一层无序列表 -->
              <li>实验一：熟悉 Windows XP 环境和操作</li>
              <li>实验二：Word 的应用</li>
            </ul>
          <li>高级语言程序设计</li>
          <li>汇编语言程序设计</li>
        </ol>
    <li>2004—2005 学年</li>
        <ol type= "a">                 <!-- 定义下一层有序列表 -->
          <li>数据结构</li>
          <li>操作系统</li>
          <li>数值分析</li>
        </ol>
</ol>
</body>
```

```
</html>
```



图 2-9 有序列表的运行示例

3. 定义列表(Definition List)

前两种列表存在一个不足，就是用户不能自定义项目的标题。在网页中充斥着大量用户希望自定义的类别信息。定义列表的出现正迎合了这种要求，同时克服了前两种列表这种不足。定义列表的出现允许用户自定义文字定义项目的名称，达到明细表能明确文本信息的作用。

定义列表与上述的无序列表和有序列表不同，不再罗列列表项目，而侧重对列表内容的说明。定义列表分成两部分：列表的描述项和列表解释项。列表的描述项确定列表的主要内容，而列表的解释项是对描述项的具体说明。定义列表的实现是通过标签 dl、dt 和 dd 组合产生的。标签 dl 表示定义一个定义列表。dt 标签定义一个描述项，而 dd 标签定义描述项的解释项。定义列表的形式如下：

```
<dl>
  <dt>描述项 1</dt>
  <dd>解释项 1</dd>
  <dt>描述项 2</dt>
  <dd>解释项 2</dd>
  ⋮
  <dt>描述项 n</dt>
  <dd>解释项 n</dd>
</dl>
```

在定义列表的每一个 dd 元素中，常常为了使内容更加生动，可以插入其他的网络资源，如图像、超链接。当然也可以向上述的其他两种列表一样嵌套包括定义列表在内各种类型

的列表。在下列的程序清单 2-9 中，展示了定义列表的应用实例，运行结果如图 2-10 所示。

程序清单 2-9：

```
<!--definitionlist.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>定义列表示意</title></head>
<body>定义列表示例->解释单词：
<dl>
  <dt>word</dt>
        <dd>词，单词</dd>
        <dd>谈话，言语</dd>
        <dd>消息，音信，谣言，传说</dd>
        <dd>承诺，诺言，保证</dd>
        <dd>命令</dd>
        <dd>口令</dd>
        <dd>[pl.]口角，争论</dd>
        <dd>格言</dd>
  <dt>web</dt>
        <dd><ol>                              <!--嵌套定义有序列表-->
                <li>蜘蛛网；蛛网状物；丝网；【纺】棉[毛]网</li>
                <li>织物；(织物的)一匹</li>
                <li>捏造的谣言，圈套</li>
                <li><p>(水鸟，蛙等的)蹼；羽瓣 一卷纸【机】连接板；金属薄片[薄条]；
                    【建】工字梁腹(部)；圆拱；【解】膜，网状组织</p>
               </li>
             </ol>
     </dd>
</dl>
</body>
</html>
```

2.1.6 表格

表格是 Web 最常见的应用。大量的数据可以通过表格来表示。表格有行和列纵横组成，每行每列中排列具体的各类型的信息，方便用户查找浏览。表格由于表达形式灵活直观，主要应用于两方面。

(1) 应用于网页中数据的表达。

(2) 应用于网页数据的展示和网页的排版设计。

在 HTML 语言中定义了表 2-4 所展示的标签来定义表格。

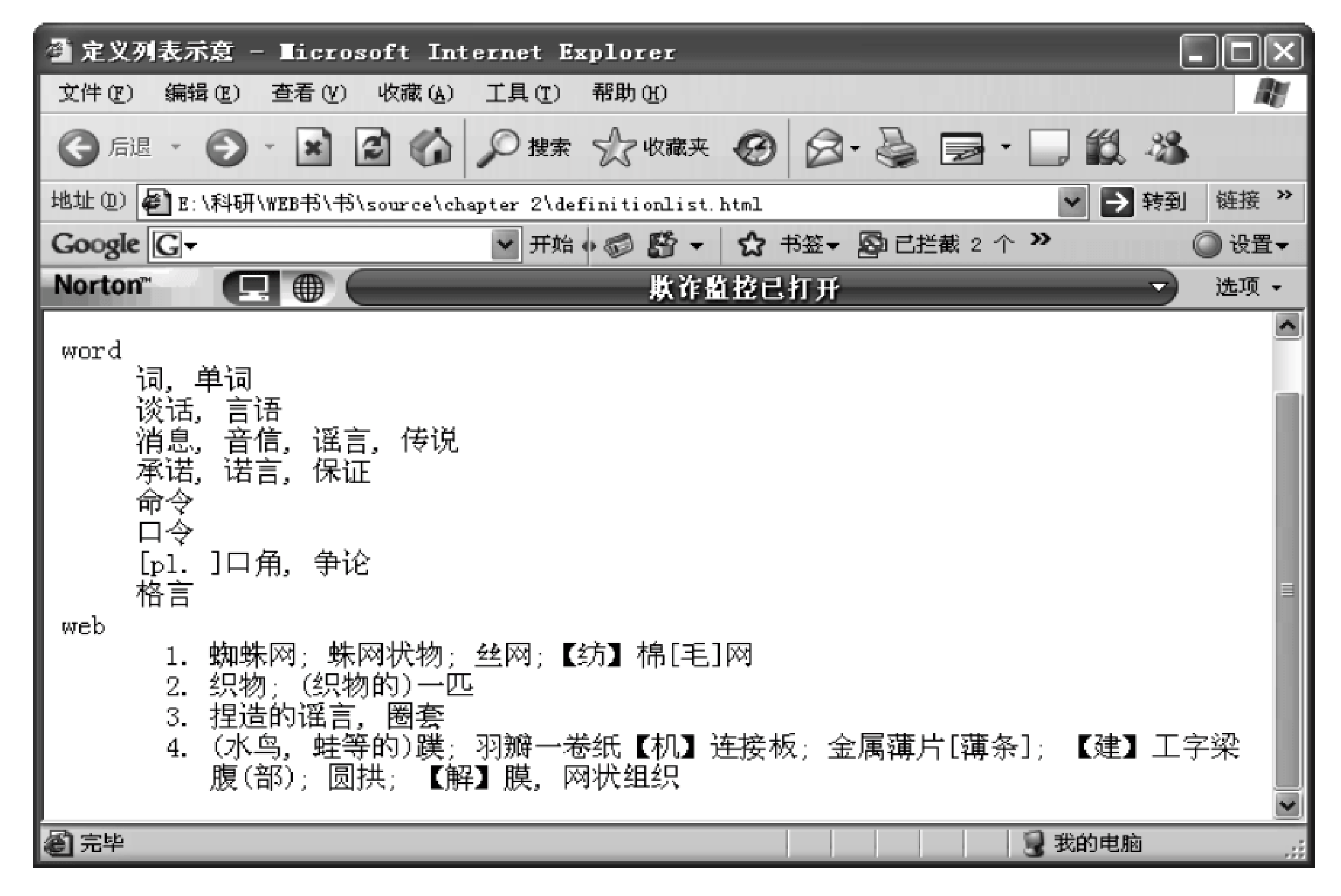


图 2-10 定义列表的运行实例

表 2-4 定义表格的常见标签

基本标签	说　　明	基本标签	说　　明
table	定义一个表格	caption	定义表格的标题
th	定义表格的单元格为标题项	thead	定义表格的表头
tr	定义表格的行	tbody	定义表格的主体
td	定义表格的每一单元格	tfoot	定义表格的表尾

一个严格表格的结构如图 2-11 所示。制作一个 HTML 表格，并不需要和图 2-11 所展示的结构一致。实际上，为了迎合更多浏览器的性能的要求，制作表格时，常常将 thead、tbody 和 tfoot 标签取消，而直接用 table 标签、tr 标签、th 标签、td 标签定义一个表格。但是，thead、tbody 和 tfoot 标签的存在会更好地体现表格的层次性，增加设计人员对网页的可读性，也符合对 HTML 4.01 标准的要求。

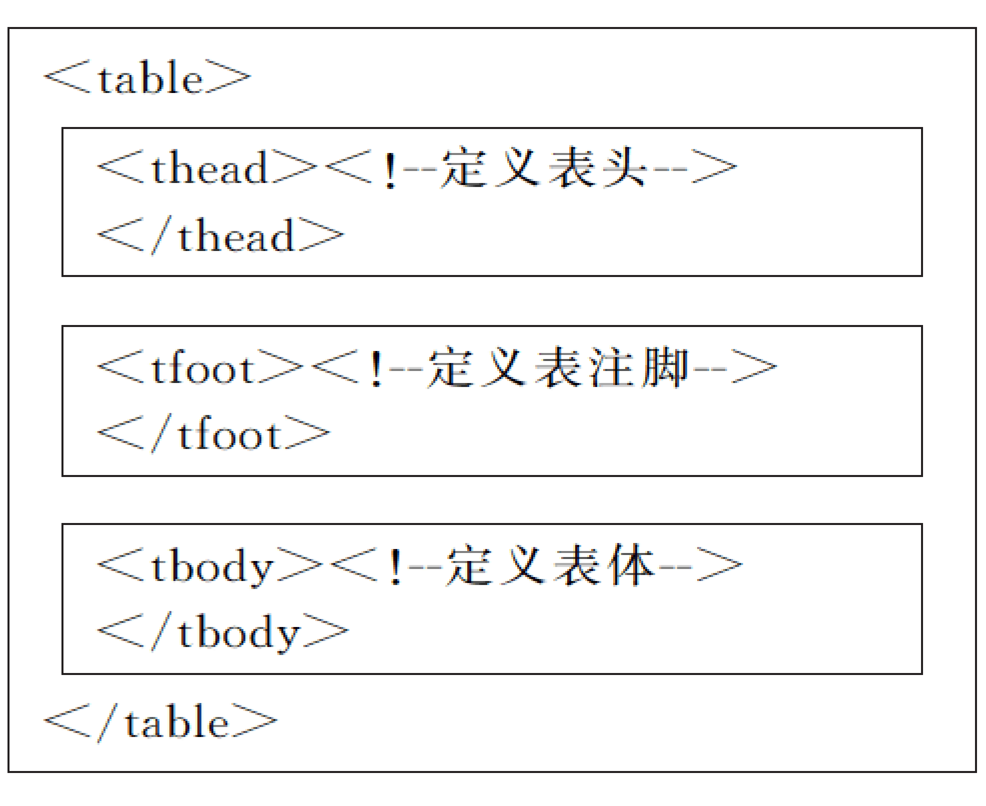


图 2-11 表格的结构

【例 2-1】 设计一个表格展示 3 名学生的期末数学与语文成绩。代码见程序清单 2-10,运行结果如图 2-12 所示。

程序清单 2-10:

```
<!--tabel1.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>成绩单表格示意</title>
</head>

<body>
    <table border="1">                          <!-- 定义表格的边界为 1-->
      <caption>期末考试成绩单</caption>          <!--定义表格的标题-->
      <thead>                                   <!-- 定义表头-->
        <tr>                                    <!-- 定义表格的一行-->
          <th>学号</th>                         <!-- 定义单元格的标题项-->
          <th>姓名</th>
          <th>语文</th>
          <th>数学</th>
        </tr>
    </thead>
<tfoot>                                         <!-- 定义表尾-->
        <tr>
          <td colspan="4">提交表格时间:2007年 12月 28日星期五</td>
          <!-- colspan属性的作用是跨越 4列-->
        </tr>
      </tfoot>
      <tbody>                                   <!-- 定义表主体-->
        <tr>                                    <!-- 定义表格的一行-->
          <td>001</td>                          <!-- 定义单元格的内容-->
          <td>张三</td>
          <td>90</td>
          <td>89</td>
        </tr>
        <tr><td>002</td>
            <td>刘四</td>
            <td>92</td>
            <td>89</td>
        </tr>
        <tr><td>003</td>
            <td>王五</td>
            <td>94</td>
            <td>99</td>
```

```
            </tr>
          </tbody>
        </table>
    </body>
    </html>
```



图 2-12　表格展示数据运行实例

程序清单 2-10 中定义了一个成绩单的表格，对于程序清单 2-10 有几点说明。

(1) 定义表格尾部 tfoot 标签，必须要使用一次 tr 标签。如果忽略 tr 标签，会导致表格尾部元素全部丢失。

(2) 在同时使用 thead、tbody 和 tfoot 标签时，可以将 tfoot 标签放置在 tbody 部分之前。这是因为浏览器渲染所有表格主体数据之前绘制 tfoot，有效地提高表格中的数据较为庞大的渲染效率。

(3) tfoot 标签的作用常常作为表格的说明，或记录表格需要统计的数据。

【例 2-2】 利用表格设计一个网页的页面结构，具体内容见程序清单 2-11，运行结果如图 2-13 所示。

程序清单 2-11：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<head>
<title>表格设计网页结构</title>
</head>
<body>
<table align="center" bgcolor="#FF9900">              <!-- bgcolor是背景属性-->
    <tr><td>主页</td>
        <td>网页设计</td>
        <td>网页资源</td>
        <td>网页制作教程</td>
      </tr>
      <tr align="center">
```

```
        <td colspan="5">
        <table bgcolor="#FFFF00">
          <tr><td>简介</td>
              <td colspan="3" rowspan="10">欢迎来到网页教学站点</td>
          </tr>
          <tr><td>主页内容 1</td></tr>
          <tr><td>主页内容 2</td></tr>
          <tr><td colspan="4">版权所有,翻版必究</td></tr>
        </table>
        </td>
    </tr>
</table>
</body>
</html>
```

图 2-13　用表格定义页面结构

它通过表格嵌套表格展示了一个具有传统的“上左右下”结构的网页,即网页的上部定义网页的导航,网页的左部定义网页的导航,网页的右部定义网页的主体以及网页的下部定义版权信息。在实际应用中,table 元素可以结合其他元素或直接利用其他元素设计网页的页面结构。

2.1.7　表单

网页中往往通过表单实现用户信息的提交。表单最主要的应用在于它提供了交互界面,帮助用户输入信息。为了实现这样的功能,表单定义了许多表单控件元素,如文本框、单选项、复选项、组合项、按钮等。在表 2-5 中定义了常见的表单标签。

表 2-5　常见的表单标签

属　　性	说　　明
form	定义表单
input	定义一个输入域
textarea	定义一个多行文本区
label	定义一个标签
fieldset	定义域,该元素将多种元素组成整体。
legend	为 fieldset 标签定义标题
select	定义下拉列表
optgroup	定义选项组
option	定义选项,常和 select 结合使用
button	定义按钮

多个表单标签的组合可以很好地实现一个具有实际意义的交互界面,最终将用户的信息提交给服务器进行处理。表单是为服务器端获取客户端的信息,为后续的信息处理提供了一种途径。一个表单的基本组成可以分成 3 个部分。

（1）表单标签。即 form 标签，通过 form 标签定义表单。具体的表单性质取决于表单属性的定义。

（2）表单域。定义表单中的输入控件，例如输入文本区控件、标签控件、按钮控件等。通过这些控件实现表单输入界面控制。

（3）表单按钮。定义表单的各种按钮，包括提交按钮 submit、复位按钮 reset 以及常规按钮。通过它们实现表单数据传送、表单数据的重置以及其他结合脚本定义动作的处理。

1. form 标签的属性

表单定义在一对＜form＞＜/form＞标签中。form 标签的主要属性见表 2-6。

表 2-6　form 标签的常见属性

属　　性	属　性　值	说　　明
method	post get	发送信息的方式
action	统一资源定位 url	发送信息的 url
name		定义表单的名称
target	_top _blank _parent _self	打开目标 URL 的方式
enctype	mine 类型	确定表单内容编码的 MIME 类型

表单中 method 属性有两种属性值 get 和 post 表示不同的发送信息的方式。get 方式是以 URL 的方式提交表单数据，即，表单的数据附着在 url 地址内容内，是表单的默认发送方式。get 发送方式存在一个问题，因为用户访问的 URL 的长度是受到限制的，如果用户发送的表单数据超过 2KB，部分内容会截取，无法传送到服务器中。还有一种特殊的情况，如果用 get 方式发送信息，且信息中附有非 ASCII 码的数据，服务器需要对信息重新编码。在这些情况下，可以转向利用 post 发送方式。post 方式是通过 HTTP post 机制，将表单数据放置在 HTML 文件头中，然后一起传送到动作 action 属性所指的 URL 地址，也就是将表单作为一个整体发送到服务器中。post 方式并不限定数据的长度，传送数据的编码形式取决于 enctype 属性指定的 MIME 类型。

2. input 标签

表单中可以定义多种表单控件，input 标签是其中最常用的控件之一。input 标签根据设定属性的不同，可以实现单行文本输入框、复选框、单选按钮、密码框等。更重要的是，通过 input 标签可以定义数据提交按钮，为数据的进一步处理提供可能。input 标签的主要属性见表 2-7。

下面通过程序清单 2-12 来了解 form 标签结合 input 标签定义表单的用法，运行结果如图 2-14 所示。

表 2-7　input 标签的常见属性

属　　性	属 性 值	说　　明
type	text	定义单行文本输入框
	checkbox	定义复选框
	password	定义密码框
	radio	定义单选按钮
	button	定义按钮
	hidden	定义隐藏文本域
	submit	定义提交按钮
	reset	定义复位按钮
name		定义表单项目的名称，submit 和 reset 除外
size	数字	定义表单域的长度
maxlength	数字	定义 text 和 password 允许输入的最大字符数
checked	checked	为 radio、checkbox、button 类型设置是否选中
value		为 button、reset、submit 定义按钮上的文本信息
disabled	disabled	除 hidden 外的其他类型，第一次装载时，用户不能执行写或选择操作

程序清单 2-12：

```
<!-- form1.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>表单实例 1</title>
</head>
<body>
<form name="frm1" action="" method="get">
<input type="text" name="text1" value="textfield" disabled="disabled" >
                                                    <!--定义文本框-->
<input type="password" name="password1" value="123456">      <!--定义密码框-->
<input type="radio" name="radio1" value="radio" checked >    <!--定义单选按钮-->
<input type="checkbox" name="checkbox1" value="checkbox" checked><!--定义复选框-->
<input type="hidden" name="hidden1" value="hidden message"> <!--定义隐藏文本-->
<input type="submit">                                        <!--提交按钮-->
<input type="reset">                                         <!--复位按钮-->
</form>
</body>
</html>
```



图 2-14　表单 input 元素运行实例

从程序清单中展示的 form1.html 可以看到，当单击提交查询内容按钮时，由于提交表单的方式采用了 get 方式，在浏览器的地址栏中表单的数据附着在 URL“?”后面，数据与数据之间用“&”连接。input 标签是一个非封闭标签，它所定义的元素在本标签内就可以实现。

3．textarea 标签

用户在和网页交互过程中，往往需要提交大量的文本信息。textarea 标签可以实现表单中多行文本的输入控制，用户可以在文本区中输入大量的文本，无须限定文本的长度。在表 2-8 中说明了 textarea 的常见属性。

表 2-8　textarea 标签的常见属性

属　性	属性值	说　明
cols	数字	确定文本区的列数
rows	数字	确定文本区的行数
disabled	disabled	第一次载入时，失去作用
name		定义文本区名字
readonly	readonly	文本区处于只读状态，用户无法执行写入修改操作

下面通过程序清单 2-13 来了解文本区的使用，运行结果如图 2-15 所示。

程序清单 2-13：

```
<!-- form2.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>表单实例 2</title>
</head>
<body>
<form>
<textarea name="textarea" rows="10" cols="30">文本区实例
```

```
</textarea>
</form>
</body>
</html>
```



图 2-15　文本区的运行结果

4．select、optgroup 和 option 标签

select、optgroup 和 option 标签往往组合在一起构成一个选择列表。select 标签表示定义一个选择列表，optgroup 标签表示定义选择列表中的选项组，而 option 是选项组中的具体选项。值得注意的是，optgroup 标签并不是构成下拉列表的主要组成。如果 optgroup 标签没有出现在下拉列表定义，表示所有 option 定义的选项默认是一个选项组的。在下列的表 2-9、表 2-10 和表 2-11 分别列出了 3 种标签的属性。

表 2-9　select 标签的常见属性

属　　性	属 性 值	说　　明
disabled	disabled	使得下拉列表失效
multiple	multiple	表示可以同时多选
name		为下拉列表命名
size	数字	定义列表可见的项目数

表 2-10　optgroup 标签的常见属性

属　　性	属 性 值	说　　明
label	文本标签	定义项目组的标签
disabled	disabled	第一次载入，选项组失效

表 2-11　option 标签的常见属性

属　　性	属 性 值	说　　明
disabled	disabled	第一次载入,项目失效
label	文本标签	定义项目标签
selected	selected	设置项目选中状态
value		定义发向服务器的值

下面程序清单 2-14 展示了表单中选择列表的示例,运行结果如图 2-16 所示。

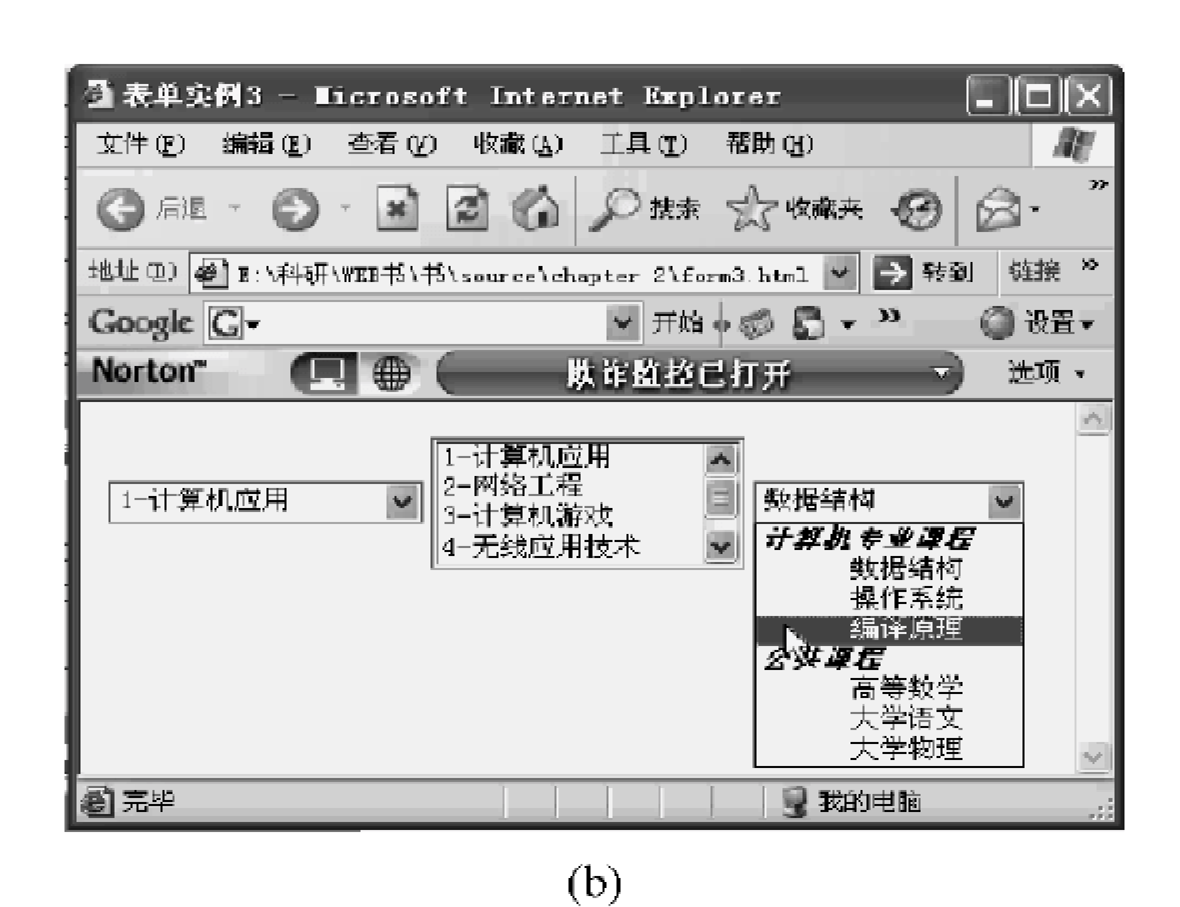
程序清单 2-14:

```
<!-- form3.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<title>表单实例 3</title>
</head>

<body>
<form>
<table>
 <tr>
<td>
 <select name="list1"><!-- 列表 1 -->
  <option value="计算机应用">1-计算机应用</option>
  <option value="网络工程">2-网络工程</option>
  <option value="计算机游戏">3-计算机游戏</option>
  <option value="无线应用技术">4-无线应用技术</option>
  <option value="计算机科学与技术">5-计算机科学与技术
  </option>
 </select>
</td>
<td>
<select name="list2" size=4><!--  列表 2 -- >
  <option value="计算机应用">1-计算机应用</option>
  <option value="网络工程">2-网络工程</option>
  <option value="计算机游戏">3-计算机游戏</option>
  <option value="无线应用技术">4-无线应用技术</option>
  <option value="计算机科学与技术">5-计算机科学与技术</option>
 </select>
 </td>
<td>
    <select name="list3"><!--列表 3 -->
      <optgroup label="计算机专业课程">
        <option>数据结构</option>
```

```
            <option>操作系统</option>
            <option>编译原理</option>
         </optgroup>
         <optgroup label="公共课程">
            <option>高等数学</option>
            <option>大学语文</option>
            <option>大学物理</option>
         </optgroup>
      </select>
      </td>
   </tr>
   </table>
   </form>
   </body>
   </html>
```

(a)



(b)

图 2-16　下拉列表的运行示例

5．fieldset 和 legend 标签

表单中通常将一些功能或应用相近的元素放置在一个域内，可以帮助用户浏览。fieldset 标签的作用是定义一个域，它将一些表单元素组合起来，组合的表单元素会显示在一个方框内。有时，为了强化域的主题或功能，常应用 legend 标签设置域的标题。在下列的程序清单 2-15 中可以说明 fieldset 和 legend 标签的应用，运行结果如图 2-17 所示。

程序清单 2-15：

```
<!-- form4.html -->
<html>
<head><title>表单实例 4</title></head>

<body>
  <form>
      <fieldset>
        <legend>用户留言本</legend>
```

```
            用户：<input type="text" name="user"><br>
            留言：<textarea rows="10" cols="20"></textarea><br>
            <input type="submit" value="发送">
            <input type="reset" value="清除">
        </fieldset>
    </form>
</body>
</html>
```



图 2-17 fieldset 和 legend 的运行实例

2.1.8 框架

框架实际上是在一个网页页面上显示多个浏览窗口。标签 frameset 和标签 frame 组合可以定义框架。其中，标签 frameset 表示定义框架的容器，标签 frame 定义框架中的子窗口。在一个框架结构中可以有多个子窗口。跟框架结构相关的标签还有 noframes。noframes 标签的作用就是避免一些版本较低的浏览器不支持框架，可以通过 noframe 将框架内该元素的内容显示出来。在下列的表 2-12 和表 2-13 中分别介绍了 frameset 和 frame 的常见属性。

表 2-12 frameset 标签的常见属性

属性	属性值	说明
cols	像素 百分比 *	定义框架的列的大小，属性的单位是像素和百分比，*表示列占据剩余的尺寸
rows	像素 百分比 *	定义框架的行的大小，其他同上

表 2-13　frame 标签的常见属性

属　　性	属 性 值	说　　明
frameborder	0 1	显示子窗口的边界与否,0 表示不显示,1 表示显示
scrolling	yes no auto	表示子窗口可以滚动
marginheight	像素	定义窗口的上下边距
marginwidth	像素	定义窗口的左右边距
name		指定窗口的名字
src	url	设置窗口链接的 url 指定的文件
noresize	noresize	设置不能修改窗口的大小
longdesc	url	定义关于窗口描述的 url。只在浏览器不支持框架时有效

通过框架结构的定义,可以有效地组织网页的页面排版结构。一个框架结构的定义形式如下所示:

```
<frameset>                          <!--  定义框架容器  -- >
  <frame>                           <!--  定义窗口  -- >
  <noframes>…<noframes>             <!--  定义浏览器不支持框架显示内容  -- >
  ⋮
<frameset>
```

框架中允许嵌套其他框架。通过这种框架间的嵌套方式,可以将一个子窗口拆分成多个子窗口,从而设计出符合用户各种要求的版面结构。在下列的程序清单 2-16 中,首先将窗口分成“上、中、下”三部分,然后将“中”部分分成“左右”两个部分。从而用框架嵌套实现复杂页面结构,运行结果如图 2-18 所示。

程序清单 2-16:

```
<!-- frameset.html -->
<html>
<head>
<title>框架实例</title>
</head>
<frameset rows="30% ,40% ,* " cols="* ">                    <!--框架定义为上中下-->
    <frame name="top" src="form1.html" scrolling="no">      <!--  定义上子窗口  -->
    <frameset row="* " cols="40% ,60% ">                    <!--  定义中子窗口  -->
      <frame name="middle_left" src="form2.html">           <!--  定义中左窗口  -->
      <frame name="middle_right" src="form3.html">          <!--  定义中右窗口  -->
    </frameset>
    <frame name="bottom" src="form4.html">                  <!--  定义下子窗口  -->
    <noframes>浏览器不支持框架,请更新浏览器的版本</noframes>
</frameset>
```

</html>



图 2-18 框架的运行实例

注意：frameset 定义的是框架容器，可以包含多个子窗口，所以 frameset 不能放置到 body 元素内。

2.1.9 图像、文本格式及其他

为了让网页具有更好更人性化的交互界面，HTML 4.01 定义了大量涉及页面显示效果的标签。主要有图像标签和涉及文本格式的各类标签。

1. 图像相关标签

为了将网页变得生动，引人入胜。HTML 提供了关于图像显示的标签。img 标签是其中一种最重要的标签。img 标签的主要作用就是将已知的图片文件插入到网页中，增加网页的展示效果。img 标签往往和其他 html 元素结合，产生美观的显示效果。例如 img 标签和链接 a 标签结合，通过图片访问到链接。又如 img 标签和 table 元素结合，可以帮助用户快速浏览信息。关于 img 标签的常见属性见表 2-14。

表 2-14 img 标签的常见属性

属　性	属　性　值	说　明
src	url	确定图片的 url 位置
alt		确定图片的文字描述
align	top\bottom\left\right\middle	确定图片排列的位置
height	像素\百分比	确定图片的高度
width	像素\百分比	确定图片的宽度
ismap		返回图像是否是服务器端图像映射
usemap		定义为客户端图像映射的一幅图像

2. 文本格式化标签

文本格式化的相关标签可以对网页的文本设置特殊的格式，如粗体，斜体等。这些关于文本格式化的标签形式简单，但功能强大，常为设计员所青睐。表 2-15 说明了常见的文本格式化标签。

表 2-15　文本格式化的常见标签

属　性	说　　明	属　性	说　　明
b	定义粗体	small	定义小写字体
i	定义斜体	em	定义强调字体
big	定义大写字体	strong	定义加重字体
sup	定义上标	sub	定义下标
ins	定义插入字	del	定义删除字
pre	预格式文本	address	定义地址

程序清单 2-17 显示了关于图像和文本格式的应用，运行结果如图 2-19 所示。

程序清单 2-17：

```
<!-- formattedpage.html -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>图像和文本格式实例</title>
</head>
<body>
<img src="image/smile.jpg">
<pre><b>粗体</b><i>斜体</i>
<sup>上标体</sup><sub>下标体</sub>
<ins>插入线</ins><del>删除线</del>
<big>大号字体</big><small>小号字体</small>
<em>强调字体</em><strong>加重字体</strong>
<address>Bei Jing Road</address>
</pre>
</body>
</html>
```

2.1.10　HTML 字符实体

在 HTML 中定义了一些特殊的字符表示了特定的含义。比如，在网页中需要表示空格不能通过空格键输入，而只能通过“ ”表达的字符来表示。像这样具有特定含义的字符称为 HTML 字符实体。字符实体具有两种形式：“& 实体名称;”和“& # 实体编



图 2-19　img 和文本格式标签的运行实例

号;”。在表 2-16 中展示了常见的字符实体。

表 2-16　常见的字符实体

字符实体	实体编号	说　明	字符实体	实体编号	说　明
		空格	<	<	<
>	>	>	&	&	&
"	"	引号	'	'	单引号

2.2　扩展超文本标记语言 XHTML

XHTML 是 EXtensible HyperText Markup Language 的缩写,表示扩展超文本标记语言。在 2000 年,W3C 联盟推出 XHTML 1.0 版。从此,XHTML 成为 W3C 标准的一员。它是在 HTML 4.0 基础上,采用 XML 形式改写而成具有良好结构的新语言。XHTML 承前启后,它既有 HTML 的语言特性,又具有 XML 语法约束,符合未来网络应用的要求。XHTML 的最终目标就是取代 HTML,更好地适用基于 XML 用户代理连接的应用要求。为了更好地适应网络应用,W3C 联盟在 2001 年推荐了 XHTML 1.1。与 XHTML 1.0 相比,XHTML 1.1 发展了 XHTML 1.0 的模块化,成为扩展 XHTML 家族的文档类型的基础。XHTML 1.1 的存在使得模块化的设计更加方便。尽管 XHTML 脱胎于 HTML 4.01,但是它并没有按照 HTML 的增加标记的轨迹发展下去。从 2002 年起,W3C 联盟制定 XHTML 2.0 草案。经过多次的修订,当前 XHTML 2.0 力求更多使用

XML,突出数据和显示分离的原则,强化模块化的支持,减少脚本的应用,使得 Web 应用的设计更加简单。

经过多年的发展,XHTML 充分体现了 HTML 和 XML 的优点。一方面,它兼容 HTML,支持 HTML 标准的标记实现文件定义;同时,它又具有 XML 的良好结构定义。由于 XHTML 是基于 XML 的,因此新的元素组只要格式规范良好并与内部结构一致,就可添加到 XHTML 的定义文档类型中。这突破了 HTML 中新元素组的添加意味着对整个定义文档类型的修改的局限。从这个角度上来说,具有 XML 特性赋予了 XHTML 的扩展性。其次,XHTML 具有良好的移植性。W3C 在 2000 年推荐了应用于受限设备的 XHTML 的最小子集 XHTML Basic。这解决了 HTML 标准在各种设备规范不一致的情况,使得 XHTML 能够服务于包括台式机、手机、PDA 等设备的网络应用。伴随着技术更新,XHTML 的可扩展性推动 XHTML 不断变革,适应于技术发展的要求。同时它的可移植性将它的应用领域不断扩大。因此,有必要对 XHTML 做一个了解。由于教材篇幅的限制,本书并不对 XHTML 新标准做详细介绍,而针对一些基本内容进行说明。

2.2.1 XHTML 文档类型定义(XHTML DTD)

在 2.1.2 节中,了解到文档类型定义 DTD 是定义一套 XML 标记的语法规则。就具体内容而言,DTD 定义了元素、子元素、属性及其取值,规定了用户在 DTD 关联的文档中可以使用什么标记,各个标记出现的顺序及其层次的关系,并定义了实体。与 HTML 一样,XHTML 具有 DTD,XHTML DTD 可以让浏览器正确理解和描述 XHTML 的语法和句法,从而达到浏览器正常执行显示网页的效果。XHTML 标准具有 3 种类型的 DTD:严格类型 DTD、过渡类型 DTD 和框架集类型 DTD。

1. 严格类型 DTD

严格类型 DTD 在网页文件中必须定义如下的语句:

```
< !DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.
dtd">
```

严格类型 DTD 要求网页格式严格,不能使用任何涉及网页表现层次的标记。一般要求声明为具有严格类型 DTD 的网页在表现层上结合层叠样式表 CSS 实现。声明严格类型 DTD 的网页充分体现数据和表示分离的原则。

2. 过渡类型 DTD

过渡类型 DTD 要求在网页文件中定义如下语句:

```
< !DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
```

过渡类型 DTD 为一些不支持层叠样式表 CSS 的网页实现提供可能。在过渡类型 DTD 中可以使用涉及表示层的标记。

3. 框架集类型 DTD

框架集类型 DTD 要求网页中声明如下语句:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

框架集类型 DTD 基本和过渡类型 DTD 一致。不同之处在于框架集类型支持 frameset 标记定义框架。而过渡类型不支持框架结构的定义标记。

2.2.2 XHTML 的语法

XHTML 是在 XML 基础上改写的 HTML 语言。XHTML 兼容 HTML 4.01，又不同于 HTML 的结构松散，具有严格的语言规则。这使得 XHTML 设计网页更加方便。现将 XHTML 的语法规则归纳成如下几点。

（1）与 HTML 不同，XHTML 文档必须通过 DOCTYPE 声明文档类型定义，这与 HTML 中 DTD 声明可以忽略形成对比。

（2）<html>标记声明中必须指定“xmlns="http://www.w3.org/1999/xhtml"”属性的 XML 名称空间。XML 名称空间标识 XHTML 文档使用的标记范围，XHTML DTD 定义的标记不与用户定义的标记或其他 DTD 中定义的标记冲突。值得注意的是，许多 xhtml 文件中并没有声明“xmlns="http://www.w3.org/1999/xhtml"”属性，但是这个属性会自动加入到代码中。

（3）XHTML 是基于 XML，XHTML 必须具有良好的格式。XHTML 标记与标记之间可以嵌套，但不能重叠。即，嵌套形如“
...”是合法的，但重叠形式如“
”是非法的。

（4）XHTML 中的所有元素必须是封闭的，如果元素本身是非封闭的，将给标记的尾端添加一个空格和“/”，以保持 XHTML 与浏览器的兼容。例如在程序清单 2-19 中，input 标记就是通过这种方式实现标记的封闭。又如，HTML 4.01 的换行标记
，在 XHTML 中就要写成
。

（5）XHTML 中的元素必须是小写的。这意味着，XHTML 中所有的标记、属性、事件等都必须写成小写形式。并且具体的属性值不能采用 HTML 简写形式，而是用双引号包括起来。比较下列程序清单 2-18 和程序清单 2-19，可以发现 HTML 网页与 XHTML 网页的明显不同。

程序清单 2-18 定义了一个 html 文件，程序清单 2-18 定义了一个 XHTML 文件，两个网页文件都可以在浏览器中显示。但是如果在程序清单 2-18 中加入 DOCTYPE 标记，声明为任何一种 XHTML DTD 类型，可以发现程序清单 2-18 的 htmldemo.html 将无法通过 XHTML DTD 的语法验证。

（6）name 属性在 XHTML 已经没有定义，取而代之的是用 id 属性来确定标记（如 a、img、applet、map、frame、form 等）的名称。

通过对 XHTML 文件语法规则的了解，可以发现 XHTML 具有和 HTML 类似的页面结构，但是 XHTML 也具有自身特点。经总结，XHTML 页面结构如下：

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>…</title>
```

```
</head>
<body>…
</body>
</html>
```

程序清单 2-18：

```
<!-- htmldemo.html -->
<!-- HTML 文件 -->
<html>
<head>
<title>xhtml 实例</title>
</head>
<body>
<form action="" method="get">
    <p>single select
    <input TYPE=radio checked>
    </p>
</form>
</body>
</html>
```

程序清单 2-19：

```
<!--xhtmldemo.html -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>xhtml 实例</title>
</head>
<body>
<form action="" method="get">
    <p>single select
    <input type="radio" checked="checked" />
    </p>
</form>
</body>
</html>
```

在介绍 XHTML 的语法规则时，已经间接地了解了 XHTML 与 HTML 之间的不同。现在对 XHTML 和 HTML 之间的语法不同做一个小结。

（1）XHTML 的标记格式良好，不能重叠，可以嵌套；HTML 语法格式要求不严格。

（2）XHTML 区分大小写，而 HTML 则不区分。

（3）XHTML 的标记必须封闭，如果是空标记，不能如 HTML 一样保持原状，必须加上

空格与/作为结束。

(4) XHTML 标记的属性必须用“=”赋予属性值,不能和 HTML 一样用属性简写代替。

(5) XHTML 中用 id 属性取代 HTML 中的 name 属性。

2.2.3 XHTML 的应用实例

XHTML 来源于 HTML 4.01 的特性,可以使用 HTML 4.01 的标记。XHTML 又具有 XML 的良好格式,从而给 XHTML 带来与 HTML 不同的特征。在本节中,将用 XHTML 实现一个“用户登录界面”来进一步了解 XHTML 的应用。见程序清单 2-20,运行结果如图 2-20 所示。

程序清单 2-20:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>用户登录</title>
</head>

<body>
  <form action="" method="post">
    <table>
      <caption>网站登录界面</caption>
      <tr>
        <td>用户名</td>
        <td><input type="text" id="username"/></td>
      </tr>
      <tr>
        <td>密码</td>
        <td><input type="password" id="password"/></td>
      </tr>
      <tr>
        <td><input type="submit" value="登录"/></td>
        <td><a href="register.html">注册</a></td>
      </tr>
    </table>
  </form>
</body>
</html>
```



图 2-20　XHTML 文件的应用实例

小　　结

HTML 语言是一门网页开发语言，它学习方便，具有较好展示网页的作用。本章从简单到复杂，从基本到高级，依次介绍了 HTML 发展历史，HTML 的页面结构、当前 HTML 技术常用的标记，以及 HTML 的字符实体。通过它们展现了如何实现网页页面的布局、超链接、列表、表格、框架以及网页多媒体的显示。虽然 HTML 具有表现的优势，但在数据定义方面存在问题。本章又从 XHTML 技术的发展情况入手，分别介绍了 XHTML 的文档类型定义、XHTML 的语法。在此过程中，对 XHTML 与 HTML 做了一个比较。帮助读者对两者的应用有一个清晰的认识。

练　习　2

1. 填空

（1）HTML 是________语言，而 XHTML 代表________语言。

（2）________方式是以 URL 提交表单数据。________方式实现表单数据作为整体发送。

（3）HTML 实现的列表有________、________和________ 3 种形式。

（4）________是字符实体。

2. 选择题

（1）阅读下列程序段，程序中存在错误的是（　）

第 1行：…

第 2行：<table>

第 3行：<thead>

第 4行：<caption>标题</caption>

第 5行：</thead>

第 6行：<tbody>

第 7行：<tr><td>单元 1</td></tr>
第 8行：</tbody>
第 9行：<tfoot><p>表格示例</p></tfoot>
第 10行：</table>
第 11行：...

A. 第 4 行　　B. 第 7 行　　C. 第 9 行　　D. 第 10 行

（2）下列程序段实现具有左右页面结构的框架，左边和右边结构宽度相等。阅读程序段，将下列空格的部分补充完整①(　)②(　)。

```
<frameset  __①__  __②__ >
    <frame src="left.html">
    <frame src="right.html">
</frameset>
```

① A cols="50% 50%"　　B. cols="50pixel，50pixel"
C. cols="50%,50%"　　D. cols="*"

② A rows="50% 50%"　　B. rows="50pixel，50pixel"
C. rows="50%,50%"　　D. rows="*"

3. 实验

用 XHTML 开发一个留言板界面。

第3章　CSS 技术

3.1　CSS 简介

单纯使用 HTML 开发的 Web 页面一经下载到客户端浏览器显示后其内容就不可变化，随着 Web 的发展，人们更多地希望能够看到动态的页面效果，因此产生了相应的 DHTML(Dynamic HTML，动态 HTML)开发技术，主要包括 CSS(Cascading Style Sheets，层叠样式表)、脚本语言、HTML 4.0 及其以上版本和支持动态效果的浏览器。

CSS 中文译为层叠样式表，它是对 Web 页面显示效果进行控制的一套标准。样式就是页面显示的文字、图片等元素的格式、风格。层叠样式也就是指当页面同一元素在显示时受到多种样式控制时，将按照一定的层次顺序处理。

我们可以使用任何的文本编辑软件来编写 CSS 文件，编写好的文件以扩展名“.css”保存，这样只需要在页面文件中调用相应的 CSS 文件即可，当然也可以将 CSS 代码直接嵌入 HTML 文件中实现对页面显示样式的控制。

CSS 扩充了 HTML 标记的属性设定，使得页面显示效果更加丰富，表现效果更加灵活，更具有动态性。同时使用 CSS 可以将页面样式定义和 HTML 文件分离，使得页面开发及维护工作更容易进行。对于使用同一种显示效果的页面，我们可以定义一个 CSS 文件，而让所有显示样式相同的页面文件都调用这个 CSS 文件，我们也可以在一个页面文件中调用多个 CSS 样式表。当我们要调整或修改样式时，只需要修改 CSS 文件即可，不需要对页面文件进行任何修改。

CSS 的主要特点可以表现在以下几个方面。

(1) 将网页的内容结构和格式控制分开。在 CSS 出现以前，网页文件中格式控制代码和页面显示内容是交错在一起的，这样对于代码的查看和修改造成了极大的不便。现在使用 CSS 可以将两者分开，这样很大程度上方便了网页设计开发人员，网页文件只负责存放页面要显示的内容，而所有的格式控制都由 CSS 文件来处理。这样不仅缩减了网页文件的体积，加快了下载显示速度，同时提高了网站的维护工作效率和网页开发效率，大大减少了工作量。

(2) 可以精确控制页面的所有元素。CSS 弥补了 HTML 的不足，提供了大量的样式属性，同时可以通过脚本代码来实现对页面元素的控制，使得对页面元素的控制更加精确。

(3) 页面显示效果更加丰富。CSS 中提供了大量的网页视觉效果，如丰富了颜色的显示，增加了大量的图像处理特效等，使得网页的显示效果更加丰富。

(4) 支持多种浏览器。现在主流的浏览器，如 IE 4.0 以上版本和 NE 4.0 以上版本都支持 CSS。

3.2　CSS 基本语法

3.2.1　CSS 的基本格式

CSS 定义的基本语法格式如下：

```
选择符{规则列表}
```

其中选择符是指要使用该样式的对象(将在 3.2.3 小节详细介绍)，它可以是一个或多个 HTML 标记、ClASS 选择符或 ID 选择符，如果为多个则使用逗号“，”进行分隔。规则列表是由一个或多个属性定义语句组成的样式规则，各语句间使用分号“；”进行分隔。属性定义语句的语法格式如下：

```
属性名：属性值
```

如：

```
h3{font-family:隶书;color:blue}
h2,h3{ font-family:宋体;color:red }
myfont{font-size:20pt}
#myfont{font-size:20pt}
```

在页面文件中对 CSS 的定义有以下几种方式。

(1) 内联样式。内联样式就是直接在页面文件中使用 HTML 标记的 style 属性。这种方式可以直接在 HTML 标记中定义该标记的显示样式，并且该样式定义只能用于这个标记。其语法格式如下：

```
<标记名 style="样式属性名 1: 属性值 1;样式属性名 2: 属性值 2;…">
```

如：

```
<p style=" font-family:宋体;color:red; font-size:10pt ">
```

下面可以通过程序清单 3-1 来详细说明，运行结果如图 3-1 所示。

程序清单 3-1：

```
<!--style_1.html 文件代码-->
<html>
<head>
  <title>样式表定义——直接定义标记的 style 属性</title>
</head>
<body>
  <p style=" font-family:宋体;color:red; font-size:10pt ">
      定义此处 &lt;p&gt;标记的 style 属性：红色宋体，字体大小为 10pt
  </p>
  <p>
      此处的 &lt;p&gt;标记不受上述 style 属性控制。
  </p>
```

```
</body>
</html>
```

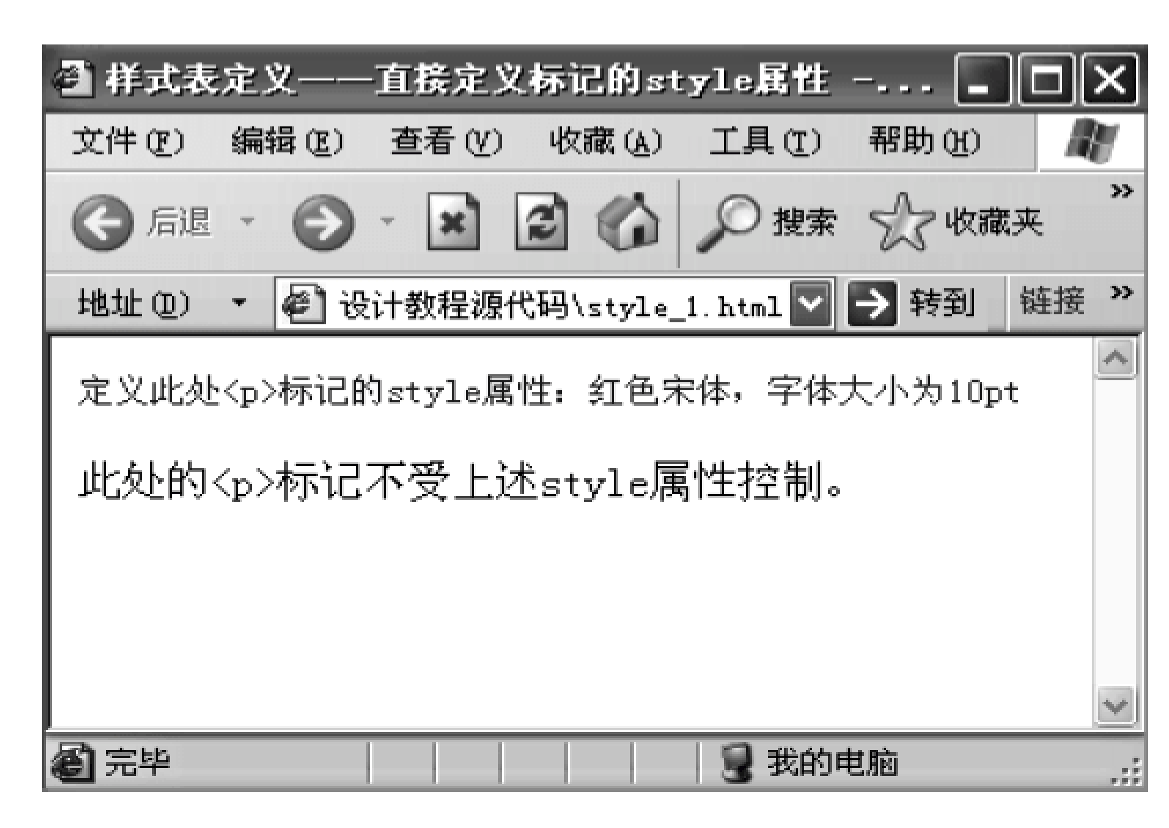


图 3-1　程序 style_1.html 运行结果

（2）在页面文件中定义内部样式表。这种方式通过<style>标记来定义样式，其语法格式如下：

```
<style type="text/css" >
<!--
    选择符 1，选择符 2，…{样式属性名 1：属性值 1;样式属性名 2：属性值 2;…}
    ⋮
-->
</style>
```

其中使用了 HTML 注释标记<!--……-->，是为了当有浏览器不支持 CSS 语句时，遇到该语句段就会忽略该段内容。下面我们可以通过程序清单 3-2 来详细说明，运行结果如图 3-2 所示。

程序清单 3-2：

```
<!--style_2.html 文件代码-->
<html>
<head>
  <title>样式表定义——内部样式表</title>
</head>
<style type="text/css">
<!--
    /*   HTML 标记   */
    h3{font-family:隶书;color:blue}

    /*   CLASS 选择符   */
    .redfont{font-family:华文彩云;color:red}
    h3.blackfont{font-family:华文行楷;color:black}

    /*   ID 选择符   */
    #myfont{font-family:黑体;color:blue;font-size:16px}
```

```
h3#hfont{font-family:华文行楷;color:red}
-->
</style>
<body>
<h3>此行文字样式为蓝色隶书</h3>
<p class="redfont">此行文字是红色华文彩云样式</p>
<p class="blackhfont">blackfont 样式仅控制 h3 标记,因此此行文字样式为默认字体</p>
<h3 class="blackfont">此行文字样式为黑色华文行楷</h3>
<p id="myfont">此行文字样式为蓝色黑体,字体大小为 16px</p>
<p id="hfont">hfont 样式仅控制 h3 标记,因此此行文字样式为默认字体</p>
<h3 id="hfont">此行文字样式为红色华文行楷</h3>
</body>
</html>
```

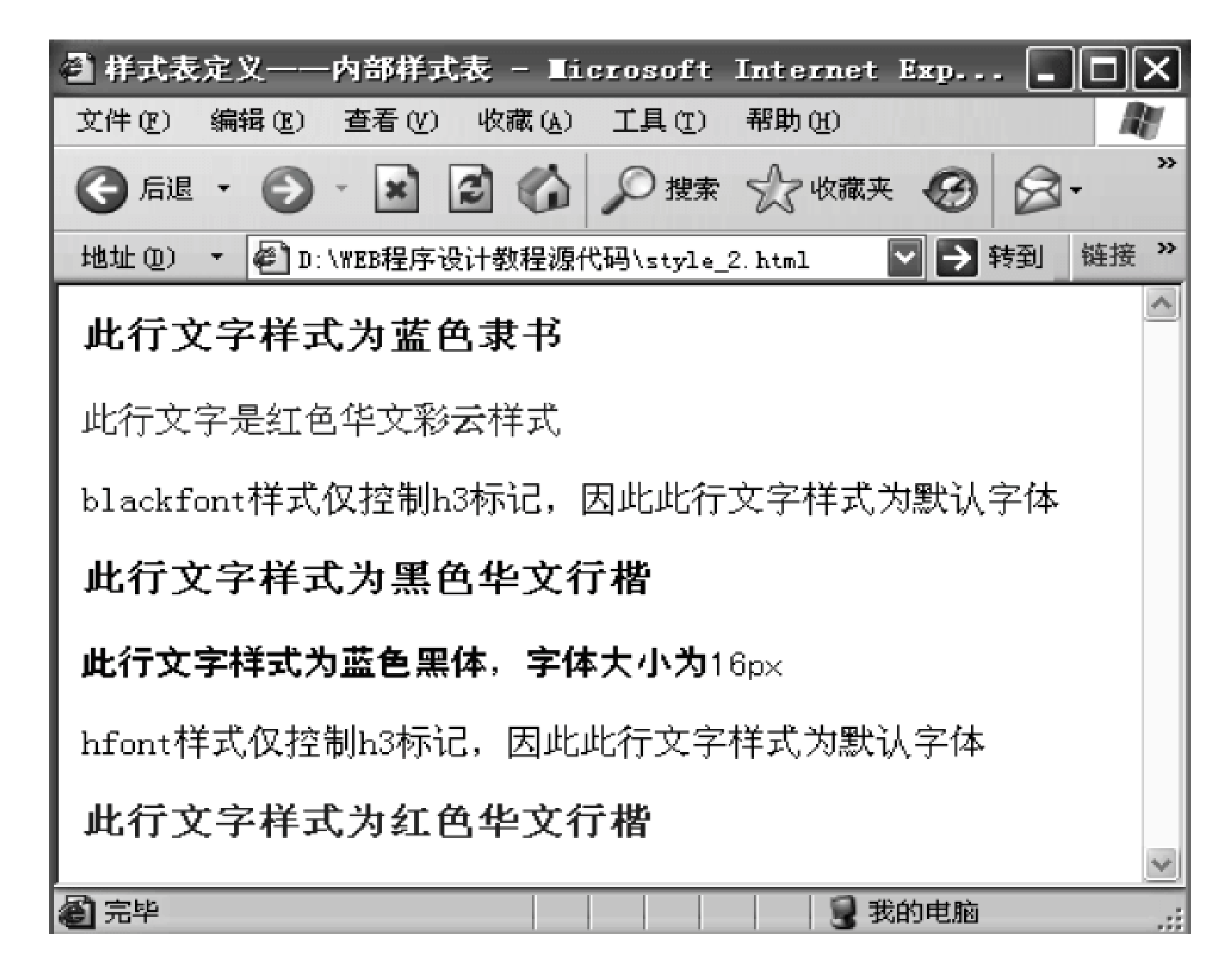


图 3-2 程序 style_2.html 运行结果

(3) 在页面文件中嵌入外部样式表。上面两种方式都是将 CSS 语句直接嵌套在页面文件中,这样这种 CSS 的定义语句只能使用于当前的页面文件。为了让多个页面文件可以共享 CSS 样式定义,可以将 CSS 语句段编写为单独的一个 CSS 文件,然后将它嵌入到页面文件中。其语法格式如下:

```
<style type="text/css" >
<!--
    @import url("外部 CSS 样式表文件名");
-->
</style>
```

其中"外部 CSS 样式表文件名"是以".css"为扩展名的 CSS 文件,如果该文件和当前页面文件处于同一目录,则直接给出文件名,否则给出其相对路径。

下面可以通过程序清单 3-3 来详细说明具体使用方法,运行结果如图 3-3 所示。

程序清单 3-3：

```
<!--style_3.html 文件代码-->
<html>
<head>
  <title>样式表定义——在页面文件中嵌入外部样式表</title>
</head>
<style type="text/css">
<!--
    @import url("mystyle.css");
-->
</style>
<body>
<h3>此行文字样式为蓝色隶书</h3>
<p class="redfont">此行文字是红色华文彩云样式</p>
<h3 class="blackfont">此行文字样式为黑色华文行楷</h3>
<p id="myfont">此行文字样式为蓝色黑体,字体大小为 16px</p>
<h3 id="hfont">此行文字样式为红色华文行楷</h3>
</body>
</html>
```

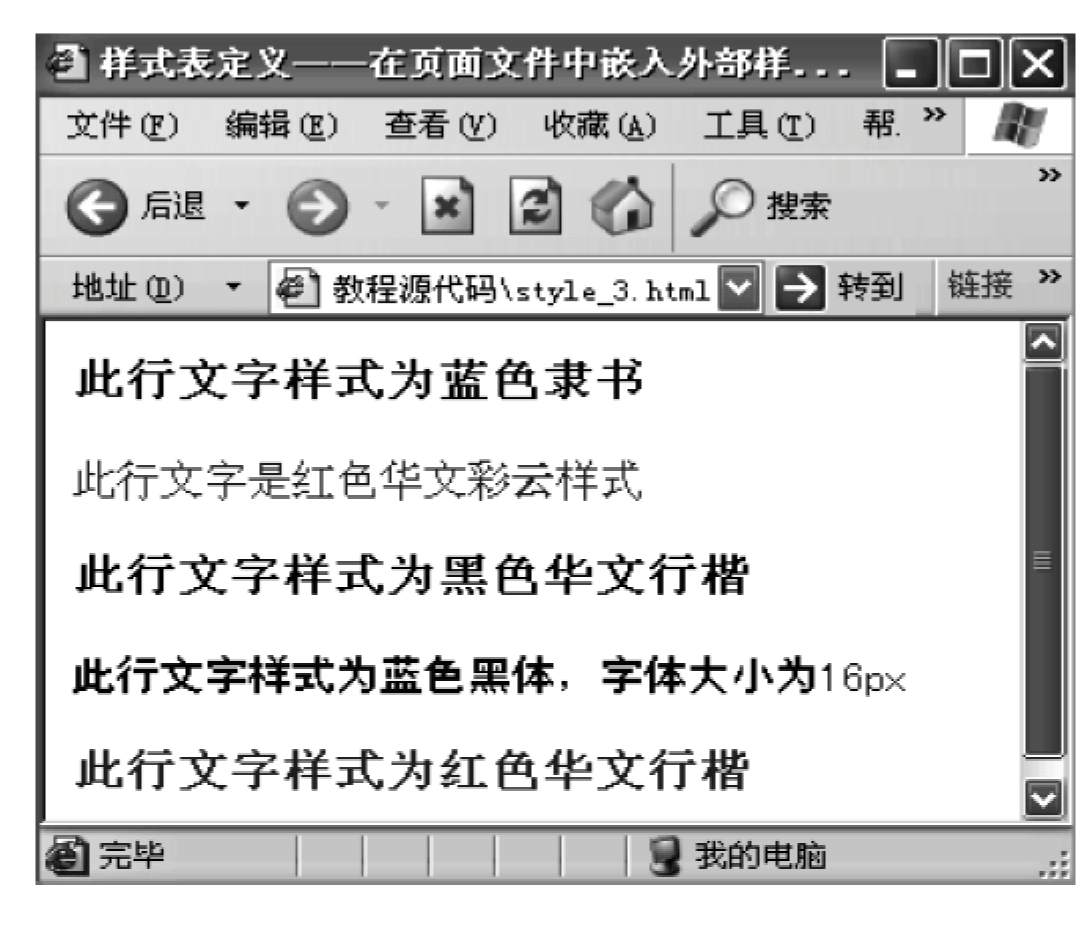


图 3-3　程序 style_3.html 运行结果

其中嵌入的外部样式表文件为 mystyle.css，该文件和 style_3.html 文件处于同一目录下。mystyle.css 文件的代码如下：

```
/* mystyle.css 文件代码 */
h3{font-family:隶书;color:blue}
.redfont{font-family:华文彩云;color:red}
h3.blackfont{font-family:华文行楷;color:black}
#myfont{font-family:黑体;color:blue;font-size:16px}
h3#hfont{font-family:华文行楷;color:red}
```

(4) 链接外部样式表。这种方式和嵌入外部样式表的方式相似，也要访问外部样式表，但是嵌入外部样式表时是将样式文件直接加载到 import 语句处，而链接外部样式表是直接

向样式文件索取样式。其语法格式如下：

```
<link type="text/css" rel=stylesheet href="外部 CSS样式表文件名">
```

下面可以通过程序清单 3-4 来说明具体使用方法，所链接的外部样式表文件就是上例中的 mystyle.css 文件。

程序清单 3-4：

```
<!--style_4.html 文件代码-->
<html>
<head>
  <title>样式表定义——在页面文件中链接外部样式表</title>
</head>
<link type="text/css" rel=stylesheet href="mystyle.css">
<body>
<h3>此行文字样式为蓝色隶书</h3>
<p class="redfont">此行文字是红色华文彩云样式</p>
<h3 class="blackfont">此行文字样式为黑色华文行楷</h3>
<p id="myfont">此行文字样式为蓝色黑体，字体大小为 16px</p>
<h3 id="hfont">此行文字样式为红色华文行楷</h3>
</body>
</html>
```

3.2.2 CSS 注释语句

在 CSS 中也可以添加注释语句来对代码进行说明。CSS 的注释语句是位于“/ * ”和“ * /”标记之间的语句内容。可以在 3.2.1 小节中的程序清单 3-2 中看到注释语句的使用。

3.2.3 CSS 选择符

CSS 选择符主要有 HTML 标记、CLASS 选择符和 ID 选择符这 3 种。它们的定义和使用方法见表 3-1。

表 3-1 CSS 选字符的定义和使用

选择符	语 法 格 式	样式使用范围说明	示 例
HTML 标记	定义语法：标记{…} 使用语法：＜标记＞	在 HTML 文件中，所有该标记包含的文本都具有定义的 CSS 样式	h3{font-family:隶书} ⋮ …
CLASS 选择符	定义语法： * .类名{…}或 .类名{…} 使用语法：＜标记 class＝类名＞	在 HTML 文件中的所有使用该类名的标记都具有定义的 CSS 样式	.my{font-size:20pt} ⋮ <h3 class=my>…</h3>
	定义语法：标记.类名{…} 使用语法：＜标记 class＝类名＞	在 HTML 文件中的所有指定该类名的该标记都具有定义的 CSS 样式	h3.my{font-size:20pt} ⋮ <h3 class=my>…</h3>

选择符	语 法 格 式	样式使用范围说明	示　　例
ID 选择符	定义语法：＃ID 名{…} 使用语法：＜标记 id＝ID 名＞	在 HTML 文件中的所有使用该 ID 名的标记都具有定义的 CSS 样式	#my{font-size:20pt} ⋮ <h3 id=my>…</h3>
	定义语法：标记＃ID 名{…} 使用语法：＜标记 id＝ID 名＞	在 HTML 文件中的所有指定该 ID 名的该标记都具有定义的 CSS 样式	h3#my{font-size:20pt} ⋮ <h3 id=my>…</h3>

可以在 3.2.1 小节中的程序清单 3-2 中看到具体的选择符的定义和使用方法。

3.2.4　样式表的层叠顺序

可以使用 CSS 语句对 HTML 标记设置不同的显示样式，但是由于 HTML 标记在使用中常常有嵌套情况出现，那么对于控制同一页面内容的嵌套标记，究竟哪一种样式起作用，是我们在本节中要说明的内容。下面我们先看一个简单的例子。

程序清单 3-5：

```
<!--cascade.html 文件代码-->
<html>
<head>
  <title>样式表的层叠</title>
</head>
<style type="text/css">
<!--
    h3{font-family:隶书;color:blue}
    h2{color:#FF0099;font-size:10px}
    p.font1{font-family:"华文行楷";color:red}
    center.font2{font-family:"宋体";color:green;font-style:italic}
    h2{font-family:黑体;font-size:20px}
-->
</style>
<body>
<h3 style=" font-family:宋体; font-size:10pt ">
    此行文字样式为蓝色宋体,字体大小 10pt
</h3>
<br>
<center class="font2">
<p class="font1">此行文字样式为红色华文行楷,斜体</p>
</center>
<br>
<h2>此行文字样式为黑体,颜色为# ff0099,字体大小 20px</h2>
</body>
</html>
```

该程序运行结果见图 3-4 所示。



图 3-4　程序 cascade.html 运行结果

从程序的运行结果我们可以看出下面几点。

(1) 在该程序中直接对<h3>标记定义了内部样式表，同时又在页面文件中使用<h3>标记的 style 属性定义了它的样式，此时后者的优先级高于前者，所以显示的文本字体样式为"宋体，字体大小 10pt"，但由于<h3>标记的 style 属性没有定义字体的颜色，所以这行文字的颜色还是由内部样式表定义的<h3>标记的颜色样式来控制，显示为蓝色。

(2) 在内部样式表中定义了 CLASS 选择符 font1 和 font2，分别控制<p>标记和<center>标记的样式。从浏览器中的第二段文字显示效果可以看出，当<p>标记嵌套在<center>标记内标注同一段文字时，最靠近这段文字的<p>标记样式 font1 优先级要高于它外层的<center>标记样式 font2。因此当它们定义了同样的属性时，起作用的为<p>标记的 font1 样式，而在 font1 中没有定义的样式属性 font-style 由于在 font2 中定义了，所以它也被继承下来同样对该段文字起作用。

(3) 在该程序中先后两次定义了<h2>标记的样式表，从浏览器中的第三段文字显示效果可以看出，当两次都定义了相同属性时第二次定义的样式优先级更高。

因此对于不同的样式定义我们可以总结如下。

(1) 直接在页面文件中使用 HTML 标记的 style 属性定义的样式优先级最高。

(2) 其他的样式定义按照在页面文件中出现的顺序，越后出现的优先级越高。

(3) ID 选择符的优先级高于 CLASS 选择符。

(4) 没有被定义样式控制的内容将使用浏览器的默认样式。

3.3　CSS 基本属性

下面将介绍常用的 CSS 的基本属性的使用方法，主要包括背景属性、文本属性、字体属性、边界属性、边框属性、边距属性、列表属性和定位属性等。

3.3.1　CSS 背景属性

CSS 中有关背景的属性、对背景颜色的设置属性和对背景图片的设置属性。

(1) background-color。用于设置背景颜色，其属性设置格式如下：

```
background-color: 颜色 | transparent
```

其中 transparent 表示透明，是该属性的默认值。颜色可以有多种表示方式。

① 使用英文名称，如 red、blue 等。

② 用 6 位十六进制数（#rrggbb）表示。

③ 用 3 位十六进制数（#rgb）表示。

④ 用 rgb（r，g，b）函数表示，其中 3 个参数的取值可为 0～255 的整数或百分比值，如：rgb(0,0,0)表示黑色，rgb(100%,100%,100%)表示白色。

（2）background-image。用于设置要加载的背景图片，其属性设置格式如下：

```
background-image: url(背景图片的地址)|none
```

（3）background-repeat。用于设置背景图片的排列方式，其属性设置格式如下：

```
background-repeat: repeat|repeat-x|repeat-y|no-repeat
```

其中 repeat 表示以并排方式排列图片，是该属性的默认值；repeat-x 表示以 x 轴方向并排排列图片；repeat-y 表示以 y 轴方向并排排列图片；no-repeat 表示不以并排方式排列图片。

（4）background-attachment。用于设置移动滚动条时背景图片位置是否固定，其属性设置格式如下：

```
background-attachment: scroll|fixed
```

其中 scroll 表示移动滚动条时背景图片随之移动，是该属性的默认值；fixed 表示移动滚动条时背景图片不动。

（5）background-position。用于设置背景图片的插入位置，其属性设置格式如下：

```
background-position: x位置值   y位置值
```

其属性值可以有 3 种取法。

① 百分比，用于描述图片距区域元素边框的百分比值。

② 数值，用具体数值描述，单位可为任意长度单位，如 em、px、in、pt、pc、cm、mm 等。

③ 关键字，包括 left、right、center、top 和 bottom。

程序清单 3-6：

```
<!--background.html-->
<html>
<head>
  <title>背景属性设置示例</title>
</head>
<style type="text/css">
<!--
    .bg1{background-image:url(./image/img1.jpg);background-repeat:no-repeat;
          background-attachment:fixed}
    .bg2{background-color:# 99CC33}
    .bg3{background-image:url(./image/img2.gif);background-repeat:repeat-x}
    .bg4{background-image:url(./image/img2.gif);background-repeat:no-repeat;
          background-position:50%  20% }
```

```
-->
</style>
<body class="bg1">
<table border="1" width="50% " align="right">
  <tr height="100" class="bg2">
    <td>背景颜色</td>
  </tr>
  <tr height="200" class="bg3">
    <td>以水平方向平铺图片</td>
  </tr>
  <tr height="200" class="bg4">
    <td>固定位置设置背景图片</td>
  </tr>
</table>
</body>
</html>
```

程序清单 3-6 运行结果如图 3-5 所示。



图 3-5　程序 background.html 运行结果

3.3.2　CSS 文本属性

CSS 中有关文本的属性主要包括以下几个。

（1）text-indent。用于设置文本的首行缩进值，默认属性值为 0（即不缩进）。其属性设置语法如下：

```
text-indent: 长度
```

其属性值用具体数值描述，单位可为任意长度单位。

（2）text-align。用于设置文本段落的水平对齐方式，其属性设置语法如下：

```
text-align: left|right|center|justify
```

该属性的取值有 4 种：

left 表示左对齐；

right 表示右对齐；

center 表示居中对齐；

justify 表示左右对齐。

（3）vertical-align。用于设置文本段落的垂直对齐方式，其属性设置语法如下：

```
vertical-align: baseline|sub|super|top|text-top|middle|bottom|text-bottom
```

其中各字段含义如下：

baseline 为该属性的默认值，表示该段落文本与其上段落文本的基线对齐；

sub 表示文字以下标方式显示；

super 表示以上标方式显示；

top 表示垂直向上对齐；

text-top 表示文字垂直向上对齐；

middle 表示垂直居中对齐；

bottom 表示垂直向下对齐；

text-bottom 表示文字垂直向下对齐。

（4）line-height。用于设置文本的行距，其属性设置语法如下：

```
line-height: normal|百分比|长度值
```

其中各字段含义如下：

normal 为该属性的默认值，表示由浏览器自动控制行距；

“百分比”表示相对于字大小 font-size 的百分比；

“长度值”用具体数值描述，单位可为任意长度单位。

（5）letter-spacing。用于设置字符的间距，其属性设置语法如下：

```
letter-spacing: normal|长度值
```

其中各字段含义如下：

normal 为该属性默认值；

“长度值”用具体数值描述，单位可为任意长度单位，我们也可将长度值设为负数，以达到紧缩字符间距的效果。

程序清单 3-7：

```
<!--text.html 文件代码-->
<html>
<head>
  <title>文本属性设置示例</title>
</head>
<style type="text/css">
```

```
<!--
    .t{text-indent:2em;text-align:center;line-height:200% ;letter-spacing:5px}
-->
</style>
<body>
<p class="t">
```

CSS 扩充了 HTML 标记的属性设定,使得页面显示效果更加丰富,表现效果更加灵活,更具有动态性。同时使用 CSS 可以将页面样式定义和 HTML 文件分离,使得页面开发及维护工作更易进行。对于使用同一种显示效果的页面,可以定义一个 CSS 文件,而让所有显示样式相同的页面文件都调用这个 CSS 文件,也可以在一个页面文件中调用多个 CSS 样式表。

```
</p>
</body>
</html>
```

程序清单 3-7 运行结果如图 3-6 所示。

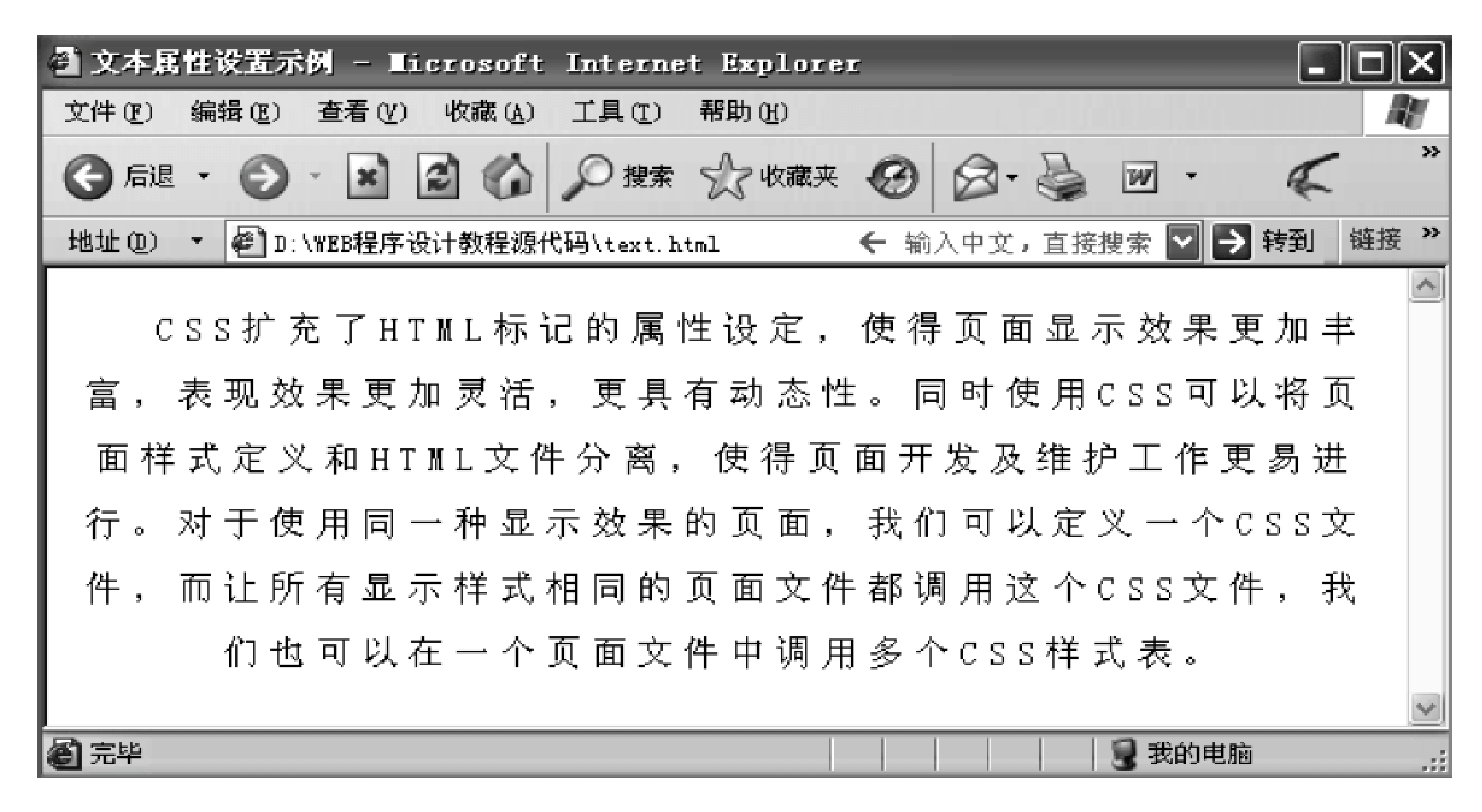


图 3-6　程序 text.html 运行结果

3.3.3　CSS 字体属性

CSS 中有关文字的字体属性主要包括以下几种。

(1) font-family。用于设置文字的字体,其属性设置语法如下:

```
font-family: 字体 1,字体 2,…
```

可以设定多种字体,用逗号“,”隔开,浏览器会按顺序依次查找系统可匹配的字体,若找到字体 1 则该文字的字体就设为字体 1,若找不到字体 1 时,再查找字体 2,依次类推。若所设的字体都找不到,则浏览器会使用默认字体。

(2) font-style。用于设置文字的效果,其属性设置语法如下:

```
font-style: normal|italic|oblique
```

其中 normal 为该属性默认值；italic 表示斜体，oblique 表示歪斜体。

（3）font-size。用于设置文字的大小，其属性设置语法如下：

```
font-size: 长度 | 关键字
```

其中“长度”用具体数值描述，单位可为任意长度单位；“关键字”包括：××-small、×-small、small、medium、large、×-large、××-large、larger 和 smaller。

（4）font-weight。用于设置字体的粗细，其属性设置语法如下：

```
font-weight: normal|bold|bolder|lighter|100~900
```

（5）font-variant。用于设置文字的变形属性，其属性设置语法如下：

```
font-variant: normal|small-caps
```

其中 normal 为默认值；small-caps 表示将中文换为较小的中文字体，而英文则转换成大写且较小的英文字体。

（6）text-decoration。用于设置文字的显示效果，其属性设置语法如下：

```
text-decoration: underline|overline|line-through|blink|none
```

其中 none 为默认值，表示没有任何效果；underline 表示给文字加下划线；overline 表示给文字加上顶部线，line-through 表示给文字加删除线；blink 表示文字闪烁。

（7）text-transform。用于设置文字的转换，其属性设置语法如下：

```
text-transform: capitalize|uppercase|lowercase|none
```

其中 none 为默认值，表示没有任何转换；capitalize 表示将每个英文单词的首字母转换为大写；uppercase 表示将所有英文字母转换为大写；lowercase 表示将所有英文字母转换为小写。

下面通过程序来说明文字属性的设置。

程序清单 3-8：

```
<!--font.html 文件代码-->
<html>
<head>
  <title>文字样式属性</title>
</head>
<style type="text/css">
<!--
    .family{font-family:隶书,宋体}
    .style{font-style:oblique}
    .variant{font-variant:small-caps}
    .weight{font-weight:900}
    .size{font-size:x-small}
    .decoration{text-decoration:line-through}
    .transform{text-transform:uppercase}
-->
</style>
```

```
<body>
<table border=1>
  <tr>
    <td class="family">样式属性设置——字体:隶书</td>
  </tr>
  <tr>
    <td class="style">样式属性设置——文字样式:oblique</td>
  </tr>
  <tr>
    <td class="variant">样式属性设置——文字变形:small-caps</td>
  </tr>
  <tr>
    <td class="weight">样式属性设置——文字粗细:900</td>
  </tr>
  <tr>
    <td class="size">样式属性设置——文字大小:x-small</td>
  </tr>
  <tr>
    <td class="decoration">样式属性设置——文字效果:line-through</td>
  </tr>
  <tr>
    <td class="transform">样式属性设置——文字转换:uppercase</td>
  </tr>
</table>
</body>
</html>
```

程序清单 3-8 运行结果如图 3-7 所示。



图 3-7 程序 font.html 运行结果

3.3.4 CSS 边界属性

CSS 中主要使用 margin 属性来控制元素边界与网页其他内容的水平和垂直间距。其

属性设置语法如下：

```
margin:距离
```

或

```
margin:距离 1  距离 2
```

或

```
margin:距离 1  距离 2  距离 3
```

或

```
margin:距离 1  距离 2  距离 3  距离 4
```

其中“距离”可以设置为具体数值或百分比形式。如果只设置一个属性值则 4 个方向均使用该值；如果设置两个属性值则垂直方向使用“距离 1”，水平方向使用“距离 2”；如果设置 3 个属性值则上方使用“距离 1”，水平左右方向使用“距离 2”，下方使用“距离 3”；如果设置 4 个属性值则上方使用“距离 1”，水平右方使用“距离 2”，下方使用“距离 3”，水平左方使用“距离 4”。

除此以外，也可以给 4 个边界单独设置具体数值，其属性设置语法如下：

```
margin-top:距离|auto
margin-right:距离|auto
margin-bottom:距离|auto
margin-left:距离|auto
```

其中 auto 为默认值，“距离”可以设置为具体数值或百分比形式。

3.3.5 CSS 边框属性

在 CSS 中可以使用 border 属性来控制元素的边框宽度、样式、颜色。其中使用 border-width 设置边框宽度，使用 border-style 设置边框显示样式，使用 border-color 设置边框颜色，每一属性都可以单独设置 4 个方向的属性值。

（1）border-width。可以使用 border-top-width、border-right-width、border-bottom-width 和 border-left-width 来设置上、右、下、左这 4 个方向的边框宽度，它们的属性值均可设为：thin|medium|thick|长度。

（2）border-style。可以使用 border-top-style、border-right-style、border-bottom-style 和 border-left-style 来设置上、右、下、左这 4 个方向的边框样式，它们的属性值均可设为：none|dotted|dashed|solid|double|groove|ridge|inset|outset。

（3）border-color。可以使用 border-top-color、border-right-color、border-bottom-color 和 border-left-color 来设置上、右、下、左这 4 个方向的边框颜色，它们的属性值均为颜色值，可以选择 CSS 的颜色表示方式进行设置。

3.3.6 CSS 边距属性

在 CSS 中可以使用 padding 来设置元素的内容与元素边框之间的距离。可以通过设置

padding 属性值控制 4 个方向的距离，也可以使用 padding-top、padding-right、padding-bottom 和 padding-left 分别设置上、右、下、左这 4 个方向的距离，其属性设置语法和 margin 属性类似，具体可参考 3.3.4 小节。

程序清单 3-9：

```
<!--margin_border_padding.html 文件代码-->
<html>
<head>
  <title>margin、border 和 padding 属性设置示例</title>
</head>
<style type="text/css">
<!--
    .margin{margin:20pt 30%  15pt 20mm}
    .border{border-width:15px 15px 15px 15px;border-color:#ff00ff;border-style:dotted}
    .padding{padding-left:10em;padding-top:10% }
-->
</style>
<body>
<table width="80%" border=1>
    <tr><td>
    <p class="margin">CSS 中主要使用 margin 属性来
    控制元素边界与网页其他内容的水平和垂直间距；
    使用 border 属性来控制元素的边框宽度、样式、
    颜色；使用 padding 来设置元素的内容与元素边框
    之间的距离。
    </p>
    </td></tr>
    <tr height="200"><td>
    <p class="border">
    CSS 中主要使用 margin 属性来控制元素边界与网页其
    他内容的水平和垂直间距；使用 border 属性来控制元
    素的边框宽度、样式、颜色；使用 padding 来设置元素
    的内容与元素边框之间的距离。
    </p>
    </td></tr>
    <tr><td>
    <p class="padding">
    CSS 中主要使用 margin 属性来控制元素边界与网页
    其他内容的水平和垂直间距；使用 border 属性来控制
    元素的边框宽度、样式、颜色；使用 padding 来设置
    元素的内容与元素边框之间的距离。
    </p>
    </td></tr>
</table>
</body>
```

```
</html>
```

程序清单 3-9 的运行结果如图 3-8 所示。

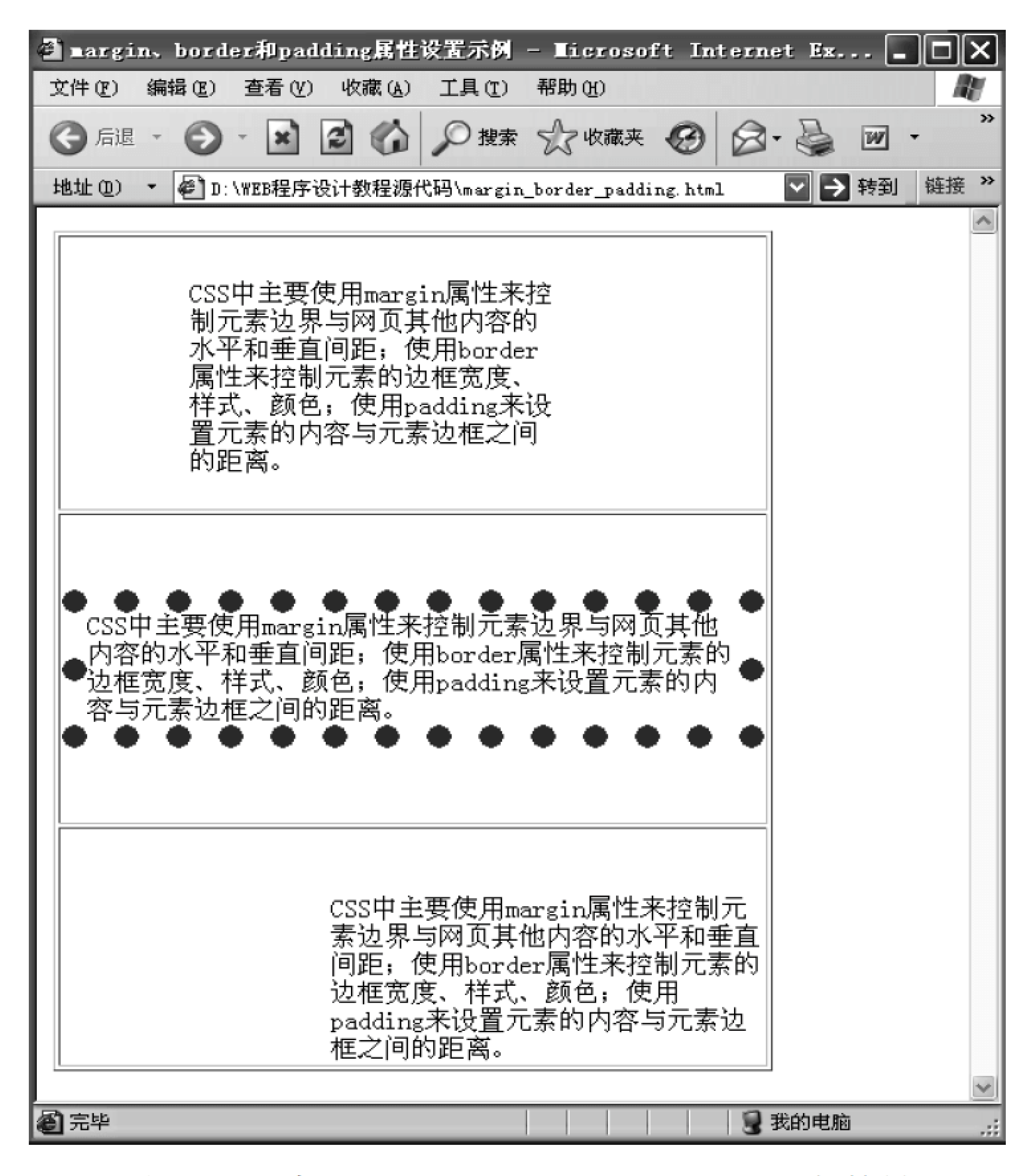


图 3-8　程序 margin_border_padding.html 运行结果

3.3.7　CSS 列表属性

CSS 中主要使用 list-style 来设置文字列表属性，控制列表的符号和位置等。其基本属性设置语法如下：

```
list-style: 符号类型,位置
```

也可以将这些属性通过使用 list-style-type、list-style-position 和 list-style-image 单独进行设定。

（1）list-style-type。用于设置列表的符号类型，其取值如下：

```
none|disc|circle|square|decimal|lower-roman|upper-roman|lower-alpha|upper-alpha
```

（2）list-style-image。用于设置使用指定图像作为列表的符号，其值为该图像的 URL 地址。语法描述如下：

```
list-style-image: url("该图像的 URL 地址")
```

（3）list-style-position。用于设置列表的项目符号的位置，其取值可为 outside 或

inside。

程序清单 3-10：

```
<!--list.html 文件代码-->
<html>
<head>
  <title>列表属性设置示例</title>
</head>
<style type="text/css">
<!--
    .list1{list-style-type:disc;list-style-position:outside}
    .list2{list-style-type:lower-alpha;list-style-position:inside}
-->
</style>
<body>
<table width="50%"border="1">
  <tr>
      <td>课程
        <ul class="list1">
        <li>数据库系统技术
        <li>面向对象程序设计
          <li>数据结构
        </ul>
      </td>
  </tr>
  <tr>
      <td>课程
        <ul class="list2">
        <li>数据库系统技术
        <li>面向对象程序设计
          <li>数据结构
        </ul>
      </td>
  </tr>
</table>
</body>
</html>
```

程序清单 3-10 运行结果如图 3-9 所示。

3.3.8 CSS 定位属性

CSS 的定位属性主要有以下几种。

(1) top。用于设置元素与浏览器窗口上方的距离。

(2) left。用于设置元素与浏览器窗口左方的距离。

(3) position。用于设置元素的位置模式，它的属性值可设为 absolute|relative|static。

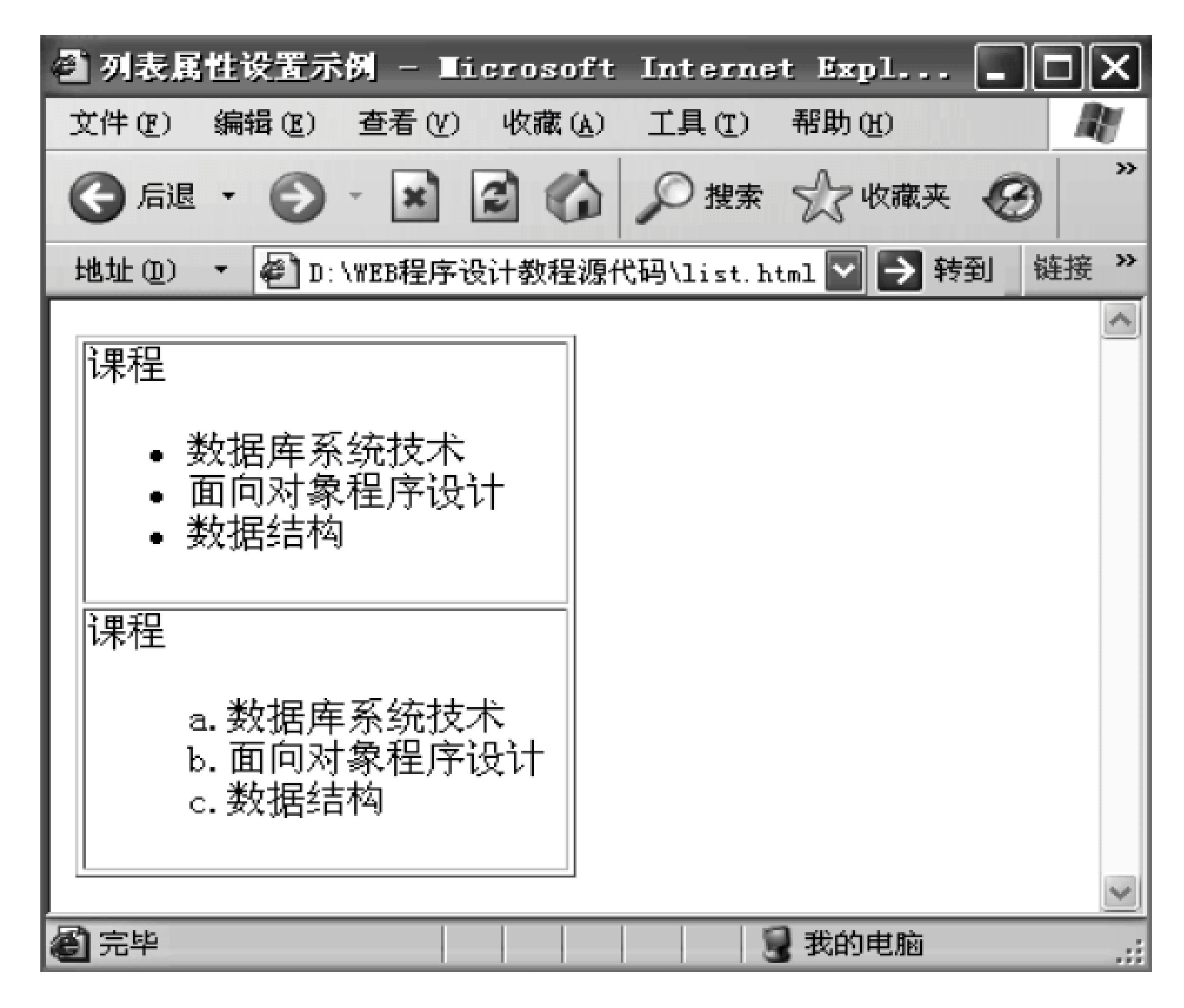


图 3-9 程序 list.html 运行结果

其中 static 为该属性默认值,表示静态位置;absolute 表示绝对位置;relative 表示相对位置。

(4) z-index。使用该属性可以将页面元素分为多层显示,产生叠加的效果。该属性的属性值为整数,可以为正整数或负整数,数值越大表示层次位置越高。

程序清单 3-11:

```
<!--position.html 文件代码-->
<html>
<head>
  <title>定位属性设置示例</title>
</head>
<style type="text/css">
<!--
    .p1{font-size:30pt;font-family:隶书;
        position:absolute;
        top:180px;left:110px;z-index:1}
    .p2{font-size:30pt;color:#FF3399;
        font-family:隶书;position:absolute;top:190px;left:120px;z-index:2}
    .p3{font-size:30pt;color:#6633FF;
        font-family:隶书;position:absolute;top:200px;left:130px;z-index:3}
-->
</style>
<body>
<div class="p1">定位属性设置示例 z-index: 1</div>
<div class="p2">定位属性设置示例 z-index: 2</div>
<div class="p3">定位属性设置示例 z-index: 3</div>
</body>
</html>
```

程序清单 3-11 运行结果如图 3-10 所示。

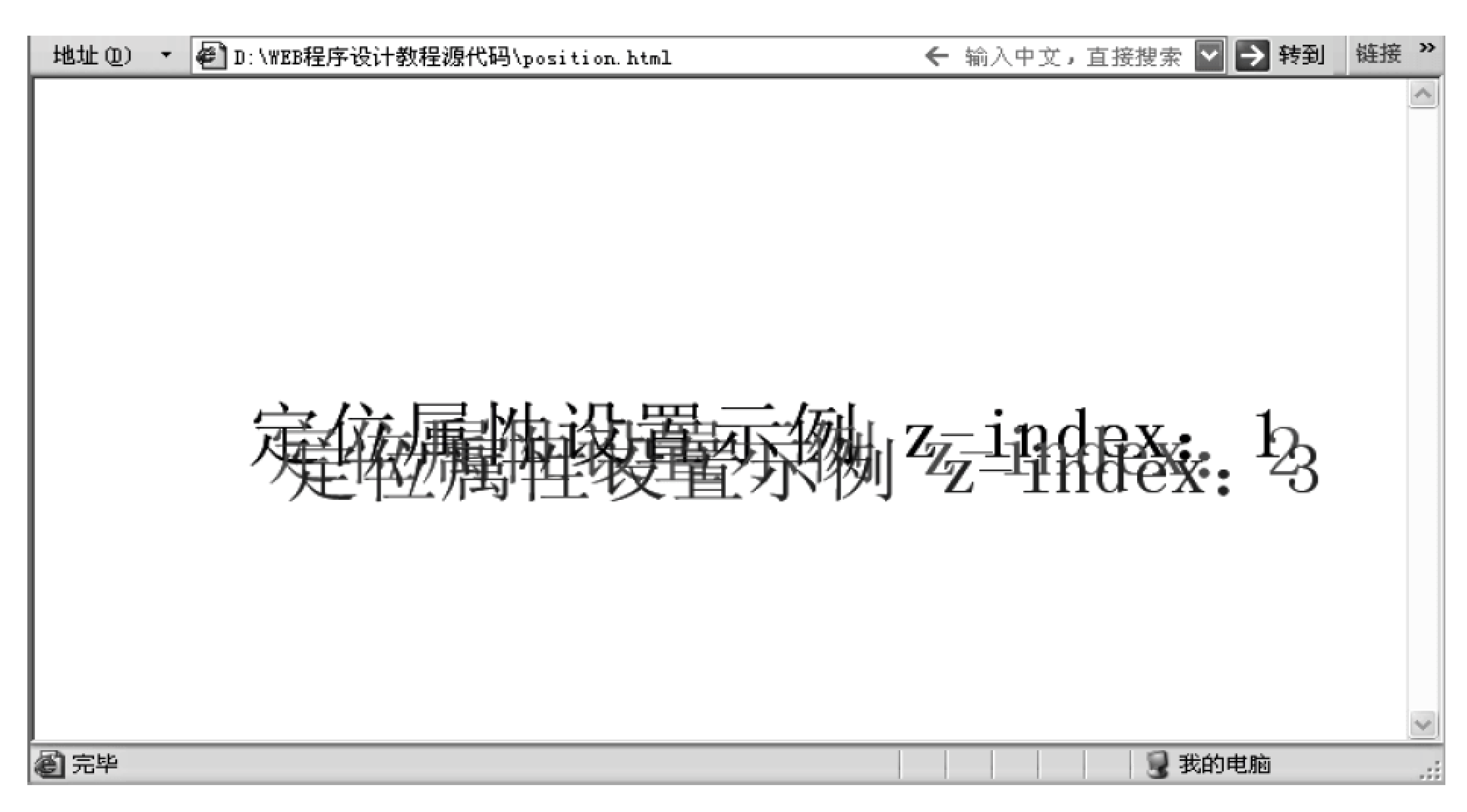


图 3-10　程序 position.html 运行结果

小　　结

本章介绍了 CSS 的基本概念和常用的属性及设置方法，主要包括以下几种：

① CSS 的基本概念；
② CSS 的定义方法；
③ 选择符的分类、定义方法和作用范围；
④ 样式表的层叠顺序；
⑤ CSS 常用的基本属性及其语法。

练　习　3

1．填空题

（1）CSS 中文译为________，它是对 Web 页面显示效果进行控制的一套标准。

（2）CSS 选择符主要有________、________和________ 3 种。

2．选择题

（1）下面可以实现给文字段落设置"2 倍行距，1.5 倍字母间距，左对齐"的 CSS 代码是________。

A.

```
p{ line-weight:200% ;
letter-spacing:150% ;
text-align:left;
}
```

B.

```
p{ line-height:200% ;
  letter-spacing:150% ;
  text-align:right;
  }
```

C.

```
p{line-height:200%;
letter-spacing:150%;
text-align:left;
}
```

D.

```
p{ line-weight:200%;
  letter-spacing:150%
  text-align:right;
}
```

(2)

```
<html>
<style type="text/css">
<!--
    p.font1{font-family:黑体;color:red;}
    center.font2{font-family:宋体;font-size:20px }
-->
</style>
<body>
<center class=font2>一个样式表<p class=font1>继承样式示例</p></center>
</body>
</html>
```

在上述代码中“继承样式示例”的文字显示样式为________。

A. 黑体、默认颜色、字体为默认大小

B. 宋体、默认颜色、字体大小 20px

C. 黑体、红色、字体大小 20px

D. 黑体、红色、字体为默认大小

3. 实验题

1. 编写 CSS 文件实现如图 3-11 所示的页面效果，其中 CSS 文件的定义分别使用下面 4 种方式完成。

(1) 直接在页面文件中使用 HTML 标记的 style 属性。

(2) 在页面文件中定义内部样式表。

(3) 在页面文件中嵌入外部样式表。

(4) 链接外部样式表。



图 3-11 实验题 1 页面效果

2. 设计一个个人主页，使用 CSS 文件控制页面文件的显示样式和风格。

第 4 章　客户端脚本语言

4.1　客户端脚本语言简介

使用 HTML 可以设计和规划页面的显示效果和内容,但是随着 WWW 技术的迅速发展,单纯使用 HTML 来编写客户端的 Web 页面程序已经无法满足用户的需求,因为 HTML 所提供的页面信息绝大多数都是静态的,通过浏览器显示给用户后页面内容固定不变,缺少动态的和交互的效果。因此各种脚本语言应运而生,它们实现了客户端页面的动态效果,提高了页面的交互性,同时使用简单、方便。

4.1.1　客户端脚本语言的作用

应用于客户端 Web 程序设计的脚本语言弥补了 HTML 语言的不足,大大增强了客户端 Web 页面的动态性和交互性,通常将脚本语言嵌入到 HTML 文档中,由浏览器直接解释执行。它的功能较为强大,可以在客户端使用脚本语言实现不同的页面动态效果,如页面显示图片的变化等,也可以利用脚本语言在客户端进行基本的输入数据的正确性检查,提高客户端与服务器端数据传输的效率和正确性。

4.1.2　常见的脚本语言

目前可以进行 Web 程序开发的脚本语言有很多种,如 JavaScript、VBScript、Perl 和 ShellScript 等,目前较为常用的是 JavaScript 和 VBScript。其中 Perl、ShellScript 通常是用于服务器端 Web 程序的开发,而 JavaScript 和 VBScript 既可以用于服务器端 Web 程序开发,也可以用于客户端 Web 程序开发。

JavaScript 具有 Java 语言的许多特性,但是应用于 HTML 页面比 Java 更简单有效,它也可以与 JSP 技术结合进行服务器端 Web 程序的开发;VBScript 则具有与 Visual Basic 几乎相同的语法结构,它还可以与 ASP 技术结合,成为服务器端的开发技术。

本章将着重介绍 JavaScript 脚本语言,使大家了解在客户端如何使用 JavaScript 进行 Web 页面程序的开发。

4.2　JavaScript 脚本语言概述

JavaScript 和 Java 有相似之处,但其实质并不一样。JavaScript 是一种嵌入 HTML 的脚本语言,它不需要编译,在客户端可以通过浏览器,如 Internet Explorer、Netscape Navigator 等解释执行。JavaScript 具有跨平台、基于对象和事件驱动的特点,同时它也具有一定的安全性。

4.2.1 初识 JavaScript 程序

下面通过一个简单的小例子先来帮助大家了解 JavaScript 脚本程序是如何嵌入到 HTML 文档中的。

程序清单 4-1：

```
<!--SimpleJS.html 文件代码-->
<html>
<head>
    <title>JavaScript 简例</title>
</head>
<body>
    <script language="JavaScript">
    alert("第一个 JavaScript 小程序");//JavaScript Appears here.
    </script>
</body>
</html>
```

在浏览器中直接打开 SimpleJS. html 文件，可以看到在页面中弹出了一个警告框如图 4-1 所示，这个警告框就是由 JavaScript 代码编写实现的。



图 4-1　SimpleJS. html 文件执行时弹出的警告框

可以使用任何的文本编辑器来编写 JavaScript 程序代码，只需要将程序嵌入到 HTML 文件中，然后通过浏览器就可以调试并运行代码。将 JavaScript 代码嵌入 HTML 文件的方法有两种。

（1）直接在 HTML 文件中使用 HTML 中的<script>...</script>标记。可以在<script>和</script>标记之间加入要书写的 JavaScript 语句，这样就将 JavaScript 程序代码直接嵌套在 HTML 文件中。其描述形式如下：

```
<script language = "JavaScript">
  JavaScript 语句段
</script>
```

（2）将 JavaScript 程序代码和 HTML 文件分别编写，并将 JavaScript 程序代码以扩展名“.js”保存，然后在 HTML 文件中通过<script>标记将指定的 JavaScript 文件导入进来。其描述形式如下：

```
<script src = "JavaScript 文件路径">
```

通常将<script>标记放置在 HTML 文件中的<head>标记或<body>标记内。如果 JavaScript 代码的内容是函数定义，一般将这类 JavaScript 代码放置在<head>标记内，因为<head>标记内的代码会在页面内容装载之前运行。

对于 JavaScript 代码的书写，还有以下几点说明。

（1）JavaScript 区分大小写。

（2）JavaScript 中使用换行符作为一条语句的结束标志。如果需要将几条语句放在同

一行书写，可以在各条语句间使用分号“;”进行分隔。

（3）JavaScript 中注释语句的描述形式有两种，可以使用“//”或“/ * ... * /”标记 JavaScript 的注释语句。要标记单行注释语句可使用“//”，则从“//”起直到换行符前的字符都被浏览器视为注释语句；要标记多行注释语句可使用“/ * ... * /”，则从“/ * ”起直到“ * /”之间的字符都被浏览器视为注释语句。

4.2.2 常见的数据类型

JavaScript 提供了 4 种基本的数据类型，分别是数值型、字符串型、布尔型和空值。

（1）数值型数据包括整数和浮点数。整数可以是十进制、八进制和十六进制，八进制数是以 0 开头，十六进制数以 0x 或 0X 开头。如：123（十进制）、035（八进制）、0x7fd（十六进制）。浮点数可由整数部分加小数部分表示，如 12.345，－3.67；也可以使用科学记数方式表示，如：3e5，1.2e-3。

（2）字符型数据是用双引号（"和"）括起来的字符串或单引号（'和'）括起来的字符。如 "hello"、'a'、"54321"等等。在 JavaScript 中也使用“\”来描述转义字符，如：'\t' 表示水平制表符、'\n' 表示换行符等。

（3）布尔型数据可以取 true 和 false 两个值。

（4）空值 null。当在 JavaScript 中试图引用一个未定义的变量就会返回空值，表示什么也没有。

4.2.3 变量

变量就是在程序中值可以改变的量。对于在程序中出现的变量我们应该明确它的命名规则、声明方法和作用域。

1. 命名规则

JavaScript 中变量名必须是以字母或下划线“_”开头，由字母、数字和下划线共同组成的字符串。在对变量命名时最好将变量名定义为能够代表该变量含义的字符串，可以便于理解。在命名变量名时要注意不能使用 JavaScript 的关键字作为变量名。在 JavaScript 中已经定义了多个关键字，常见的如：char、int、boolean、long、float、double、true、false、for、while、do、break、continue、return、if、else、var、new、this 和 case 等。

2. 声明方法

在 JavaScript 中变量的声明方法有两种。

（1）使用关键字“var”声明变量。描述形式如下：

```
var 变量名
```

或

```
var 变量名=变量值
```

如：

```
var speed
var a=10
```

(2) 不使用关键字“var”，直接使用赋值语句声明变量。描述形式如下：

变量名=变量值

如：

```
str="this is an apple"
num=12
```

在 JavaScript 中变量在声明时可以不用说明它的类型，而是在变量使用时根据该变量的值来确定其类型。另外，虽然在变量声明时关键字“var”是可选的，但是在变量声明时不能既不用关键字“var”，又不给该变量赋初值。

3. 作用域

JavaScript 中变量也可以分为全局变量和局部变量两类。全局变量定义在所有函数体外，对任何函数可见；局部变量是定义在某一函数体内，只对该函数可见。

4.2.4 常量

常量就是在程序中值保持不变的量。在 JavaScript 中常量可以是整型、布尔型、字符型，等等，如 123、21.34、"this is an apple"等。

4.2.5 运算符

JavaScript 中运算符可以分为赋值运算符、算数运算符、逻辑运算符、关系运算符、字符串运算符和位运算符 7 类。

(1) 赋值运算符。赋值运算符由基本赋值运算符“＝”和复合赋值运算符“＋＝”、“－＝”、“＊＝”、“/＝”和“%＝”组成。具体使用方法和含义见表 4-1。

表 4-1 赋值运算符使用方法

运算符	使用方法	含　义
＝	a＝b＋100	将赋值号“＝”右边表达式的值符给赋值号左边的变量
＋＝	a＋＝b	相当于 a＝a＋b
－＝	a－＝b	相当于 a＝a－b
＊＝	a＊＝b	相当于 a＝a＊b
/＝	a/＝b	相当于 a＝a/b
%＝	a%＝b	相当于 a＝a%b

(2) 算数运算符。算数运算符由＋、－(二元运算)、＊、/、%、＋＋、－－、－(一元运算)组成。其操作数和操作结果都是数值型数据。具体使用方法和含义见表 4-2。

(3) 逻辑运算符。逻辑运算符由 &&、|| 和! 组成。其中 && 和 || 是二元运算符，! 是一元运算符，其操作结果为布尔型数据。具体使用方法和含义见表 4-3。

(4) 关系运算符。关系运算符由＝＝、!＝、＞、＞＝、＜ 和 ＜＝ 组成。其操作结果是布尔型数据。具体使用方法和含义见表 4-4。

表 4-2　算数运算符使用方法

运算符	使用方法	含　　义
＋	a＋b	加法
－(二元运算)	a－b	减法
*	a * b	乘法
/	a/b	除法
＋＋	a＋＋	自增,相当于 a＝ a＋1
－－	a －－	自减,相当于 a＝ a－1
－(一元运算)	－a	取负

表 4-3　逻辑运算符使用方法

<table>
<tr><th>运算符</th><th>使用方法</th><th>含　　义</th></tr>
<tr><td>&&</td><td>a&&b</td><td>当运算符“&&”左右的操作数值均为 true 时,运算结果为 true,否则运算结果为 false</td></tr>
<tr><td>||</td><td>a||b</td><td>当运算符“||”左右的操作数值有一个为 true 时,运算结果为 true,否则运算结果为 false</td></tr>
<tr><td>!</td><td>!a</td><td>当 a 值为 true 时,运算结果为 false,当 a 为 false 时,运算结果为 true</td></tr>
</table>

表 4-4　关系运算符使用方法

运算符	使用方法	含　　义
＝＝	a＝＝b	若变量 a 和 b 值相等,运算结果为 true,否则为 false
!＝	a!＝b	若变量 a 和 b 值不相等,运算结果为 true,否则为 false
＞	a＞b	若变量 a 大于 b,运算结果为 true,否则为 false
＞＝	a＞＝b	若变量 a 大于或等于 b,运算结果为 true,否则为 false
＜	a＜b	若变量 a 小于 b,运算结果为 true,否则为 false
＜＝	a＜＝b	若变量 a 小于或等于 b,运算结果为 true,否则为 false

(5) 字符串运算符。字符串运算符为“＋”,表示字符串的连接运算。若运算符左右两个操作数均为字符串类型则直接进行字符串连接运算,若有一个操作数不是字符串类型,则先将该操作数转换为字符串类型再进行运算。如:

```
var str="he"+"llo"      //str的值为字符串"hello"
var s="12"+34           //s的值为字符串"1234"
```

(6) 位运算符。位运算符由 &、|、~ 、^ 、＜＜ 和 ＞＞ 组成。位运算符均为二元运算符,它将操作数视为二进制数进行相关运算,返回值为数值型数据。具体使用方法和含义见表 4-5。

表 4-5　位运算符使用方法

运 算 符	使 用 方 法	含　　义
&	a&b	按位与
｜	a！＝b	按位或
～	a＞b	按位取反
^	a＞＝b	按位异或
＜＜	a＜＜b	左移
＞＞	a＞＞b	右移

4.2.6　对象和数组

数组是若干同类型数据的集合。在 JavaScript 中没有定义基本的数组类型，可以通过它定义的内置对象 Array 来创建数组对象。

1. 对象

(1) 基本概念。要了解 Array 对象的使用方法，首先应该知道 JavaScript 中对象的相关知识。和 Java 不同，JavaScript 是基于对象的，之所以称它基于对象而不是面向对象，是因为 JavaScript 并不能提供抽象、继承、重载等面向对象的基本特性，但是可以在 JavaScript 中定义对象，也可以使用其内置对象来实现我们需要的操作。

在 JavaScript 中对象包含属性和方法两部分。属性就是对象的物理信息的描述，而方法指该对象可以进行的操作。要使用对象，必须确保该对象已经存在，否则会出现错误。在 JavaScript 中对象的引用可以通过 3 种方式进行：引用 JavaScript 的内置对象、引用浏览器对象和自定义对象。

(2) 对象的属性和方法的访问方式。对对象中属性的访问可以通过“对象名.属性名”的方式进行，同样，要调用对象中定义好的方法可以通过“对象名.方法名”的形式实现。如 student 是一个定义过的对象，它包含 name、stunum 和 age 这 3 个属性，当我们要给这三个属性赋值时可以通过以下语句实现：

```
student.name="John"
student.stunum="12345 "
student.age=20
```

另外也可以将对象的属性看作是一个属性数组，因此可以通过访问数组下标的形式访问对象的属性，数组下标可以用数字或者属性名表示。如对于上述 student 对象属性的赋值也可以通过以下语句实现：

```
student[0]="John"
student[1]="12345"
student[2]=20
```

或

```
student["name"]="John"
```

```
student["stunum"]="12345 "
student["age"]=20
```

（3）自定义对象。在 JavaScript 中要自定义对象，可以使用关键字 function 来声明。其语法形式如下：

```
function 对象名(属性列表){
this.属性 1=参数 1
this.属性 2=参数 2
⋮
this.方法 1=函数 1
this.属性 2=函数 2
⋮
}
```

如要定义一个 student 对象，它的属性为 name、stunum 和 age，可以通过以下语句实现：

```
function student(Name, StuNum, Age){
this.name=Name
this.stunum=StuNum
this.age=Age
}
```

当定义好后我们就可以用 student 创建具体实例了。如：

```
var s=new student("John","12345",20)
```

2. 数组对象

在 JavaScript 中可以通过使用其内置对象 Array 来创建数组对象。

（1）创建数组对象。创建具体数组对象实例的方法如下：

```
var 数组名=new Array()
```

或

```
var 数组名=new Array(数组长度)
```

其中数组名是一个变量，可按照变量命名的要求来命名，数组的长度就是数组能够存放元素的个数。若定义时没有指定数组长度，则数组长度根据后面具体使用时确定。数组的下标从 0 开始。

（2）Array 对象的属性和方法。Array 对象常用的属性有 length，用来描述数组的长度；常用的方法有 join()、reverse()和 sort()方法。join()方法用于将数组的所有元素连接成字符串返回，reverse()方法用于将数组的所有元素逆序，sort()方法用于对数组元素按字母顺序进行排序。

下面通过程序清单 4-2 来说明数组的创建和使用方法，程序运行结果见图 4-2。

程序清单 4-2：

```
<!--Array.html 文件代码-->
<html>
```

```
<head>
<title>Array对象使用示例</title>
<script language= "JavaScript">
    function Student(Name,StuNum,Age){
        this.name=Name
        this.stunum=StuNum
        this.age=Age
        this.showStuInfo=showStuInfo
    }
    function showStuInfo(Name){
        document.StuForm.StuName.value=Name
        document.StuForm.StuNum.value=this.stunum
        document.StuForm.StuAge.value=this.age
    }
    function displayArray(arr){
        document.writeln("  ")
        document.writeln("数组共有 "+arr.length+" 个元素,分别为: "+arr.join())
        document.writeln("<br>")
    }
</script>
</head>
<body>
<script language="JavaScript">
    var Students=new Array()  //创建数组
    //创建数组内的对象并赋初值
    Students[0]=new Student("Mike","s001",20)
    Students[1]=new Student("Jenny","s002",21)
    Students[2]=new Student("Emma","s003",20)
    var Students2=new Array()
    Students2[0]="Michale"
    Students2[1]="Ben"
    Students2[2]="Paul"
    Students2[3]="Cherry"
</script>
<br>
<center>
<h2>Array对象的创建和使用示例</h2>
<br>
<font color="#660033" size="4">
单击下面的按钮,选择要查看的学生信息
</font>
<br><br>
<input type="button" value="Mike" onClick="Students[0].showStuInfo('Mike')">
<input type="button" value="Jenny" onClick="Students[1].showStuInfo('Jenny')">
<input type="button" value="Emma" onClick="Students[2].showStuInfo('Emma')">
```

```
<br>
<form name="StuForm">
姓名：<input type="text" name="StuName"><br>
学号：<input type="text" name="StuNum"><br>
年龄：<input type="text" name="StuAge">
</form>
</center>
<hr size="1">
<br><br>
<h2 align="center">Array对象的常用属性和方法使用示例</h2>
<h3>
<script language="javascript">
    document.writeln("使用 length属性和 join()方法：<br>")
    displayArray(Students2)
    document.writeln("<br>")
    document.writeln("调用 reverse()方法：<br>")
    Students2.reverse()
    displayArray(Students2)
    document.writeln("<br>")
    document.writeln("调用 sort()方法：<br>")
    Students2.sort()
```

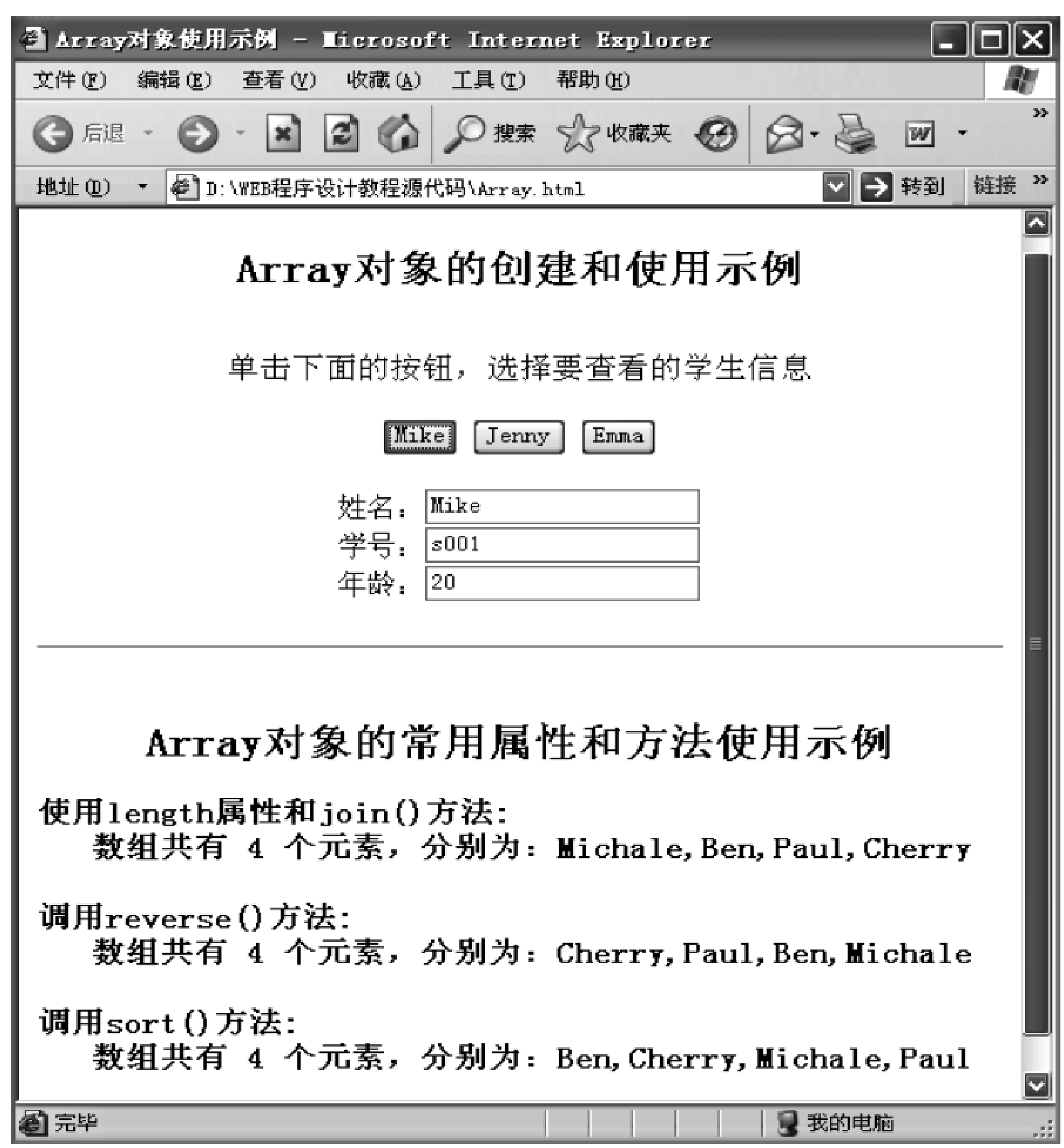


图 4-2　程序 Array.html 运行结果

```
    displayArray(Students2)
    document.writeln("<br>")
</script>
</h3>
</body>
</html>
```

4.3 JavaScript 的控制流程

程序控制流程使得程序能够按照某种指定的顺序运行。在 JavaScript 中也包含顺序、分支和循环三种控制结构。顺序结构是按照程序语句出现的先后顺序依次执行;分支结构包括条件语句 if…else 和选择语句 switch…case;循环结构包括 for 循环和 while 循环。

4.3.1 条件语句 if…else

if 条件语句的基本语法形式如下:

```
if(条件表达式){
     语句段 1
     }
else  {
     语句段 2
     }
```

若条件表达式结果为 true 则执行语句段 1,否则执行语句段 2。如果程序不需要处理条件不成立的情况,可以不写 else 语句段。当然 if 语句也可以进行嵌套,如:

```
if(条件表达式 1){
    语句段 1
    }
else if(条件表达式 2){
    语句段 2
    }
else if(条件表达式 3){
    语句段 3
    }
⋮
else {
    语句段 N
    }
```

4.3.2 选择语句 switch…case

switch 语句的基本语法形式如下:

```
switch(表达式){
```

```
case 值 1: 语句段 1
            break
case 值 2: 语句段 2
            break
⋮
default: 语句段 N
            break
}
```

switch 语句首先计算表达式的值，然后根据该值匹配相应的 case 值，决定执行哪个语句段。

4.3.3 计数循环语句 for

for 语句的基本语法形式如下：

```
for(初始化表达式; 条件表达式; 增量表达式){
循环体语句段
}
```

其中初始化表达式在整个循环体执行之前执行一次；条件表达式用来设置循环体的执行依据，通常是一个逻辑表达式，每次执行循环体之前要先检查条件表达式，若为 true 则执行循环体一次，否则离开循环体执行后继语句；增量表达式在每次循环体执行后执行，通常用来改变计数变量的值。

在 JavaScript 中除了 for 语句外，while 语句也可以实现循环。while 语句有两种。

（1）while 语句。其基本语法形式如下：

```
while(条件表达式){
    循环体语句段
}
```

这种 while 语句在每次执行循环体前先检查条件表达式是否成立，当表达式为 true 时执行循环体，否则离开循环体执行后继语句。

（2）do…while 语句。其基本语法形式如下：

```
do {
    循环体语句段
} while(条件表达式)
```

do…while 循环在每次执行完循环体后才计算条件表达式的结果是否为 true，如为 true 则继续执行一次循环体，否则离开循环体执行后继语句。

4.3.4 循环语句 for…in

for…in 语句的基本语法形式如下：

```
for(变量名 in 对象名){
    循环体语句段
}
```

for…in 语句是用来对已知对象的所有属性进行循环操作的循环语句。它不需要使用计数器来控制循环次数,也无须知道对象的属性个数就可以进行相关操作。

下面通过一个例子来说明如何使用 for…in 语句访问对象的所有属性值,程序代码见程序清单 4-3,运行结果见图 4-3。

程序清单 4-3:

```
<!--for_in.html 文件代码-->
<html>
<head>
<title>for…in语句使用示例</title>
<script language="JavaScript">
    function Student(Name,StuNum,Age){
        this.name=Name;
        this.stunum=StuNum;
        this.age=Age;
    }
    function displayProp(obj){
        var property;
        for(property in obj) //使用 for_in语句访问对象 obj 中的所有属性
            document.writeln(obj[property]+" ,");
        document.writeln("<br>");
    }
</script>
</head>
<body>
<script language="JavaScript">
    var Students=new Array();
    Students[0]=new Student("Mike","s001",20);
    Students[1]=new Student("Jenny","s002",21);
    Students[2]=new Student("Emma","s003",20);
</script>
<br>
<center>
<script language="javascript">
    var i=0;
    while(i<3){
        displayProp(Students[i]);
        i=i+1;
    }
</script>
</center>
</body>
</html>
```



图 4-3 程序 for_in.html 运行结果

4.3.5　with 语句

当需要访问某个对象的若干属性和方法时，通常要在属性和方法前加上该对象名，这样代码书写较为繁琐。JavaScript 中提供了 with 语句来简化对对象的属性和方法的访问，其语句格式如下：

```
with(对象名){
    语句段
}
```

对于上例 for_in.html 程序中的函数 displayProp(obj)可以使用 with 语句替换 for…in 语句，代码段如下：

```
function displayProp(obj){
        var property;
        with(obj) {//使用 with 语句访问对象 obj 中的所有属性
            document.writeln(name+" ,");
            document.writeln(stunum+" ,");
            document.writeln(age);
        }
            document.writeln("<br>");
    }
```

4.4　JavaScript 的函数

在 JavaScript 可以通过定义函数来封装在程序中需要重复使用的功能模块代码，这样可以使程序结构清晰，并且易于维护。函数要先定义后使用，通常将函数定义放在 HTML 文件的<head>标记内，然后可以在<body>标记内调用已经定义好的函数。

4.4.1　函数的定义

在 JavaScript 中函数的定义要使用关键字 function，其基本的语法形式如下：

```
function 函数名 (参数 1, 参数 2, …, 参数 n){
    函数体语句段
    return 表达式
}
```

其中参数不是必须的，可以根据需要进行定义，如果函数不需要返回任何结果，也可以不写 return 语句。

4.4.2　函数的调用

通过 function 定义了函数后，就可以调用该函数执行已经定义好的操作，函数的调用方法很简单，只需要给出已定义的函数名和要传递到函数中的参数值就可以。在 JavaScript 中调用函数时，当给出的实参个数和函数定义的参数数目不相同时，并不会出现错误。下面

通过程序清单 4-4 给出的程序来说明如何在 HTML 文件中进行函数定义和调用，以及如何处理调用函数时参数多于函数定义时的参数个数的情况，运行结果如图 4-4 所示。

程序清单 4-4：

```
<!--Function.html 文件代码-->
<html>
<head>
<title>函数定义使用示例</title>
<script language="JavaScript">
    function max(a,b){
        if(a>b)   return a;
        else      return b;
    }
    //max2()函数可处理调用函数时参数比定义的参数个数多的情况
    function max2(){
        var maxNumber;
        var number=max2.arguments.length;  //获取实际参数的个数
        for(var i=0;i<number-1;i++){
            if(max2.arguments[i]>max2.arguments[i+1])
                maxNumber=max2.arguments[i];
            else
                maxNumber=max2.arguments[i+1];
        }
        return maxNumber;
    }
</script>
</head>
<body>
<center>
<script language="JavaScript">
    document.writeln("12 和 23 中较大的数为："+max(12,23));    //普通函数调用
    document.writeln("<br><br>");
      //实际参数个数比函数定义的参数个数多时的函数调用
    document.writeln("13、6、38、27 中最大数为："+max2(13,6,38,27));
</script>
</center>
</body>
</html>
```

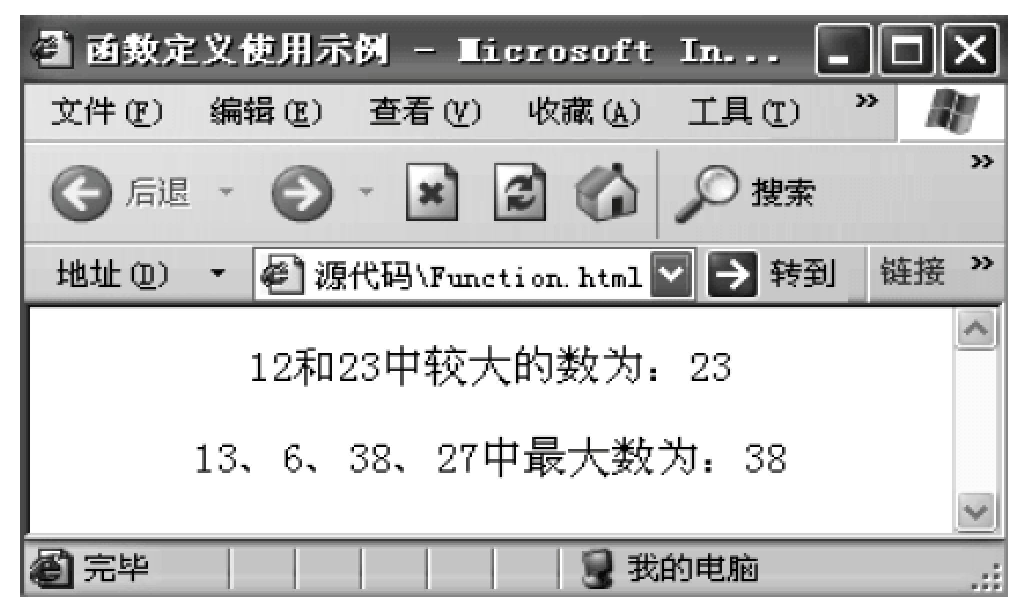


图 4-4　程序 Function. html 运行结果

在程序 Function. html 中使用了系统变量 arguments. length 来获取实际传递到函数 max2( ) 的实参个数，从而当实际参数多于函数定义的参数个数时，max2( ) 函数也能够求出所有参数中的

最大值,并不会影响函数的运行。

4.5 JavaScript 的事件处理

JavaScript 具有事件驱动的特点。事件就是用户对计算机所进行的相关操作,如鼠标的移动、鼠标键的单击等。

4.5.1 事件处理

JavaScript 中的事件可以理解为用户和 Web 页面的交互操作,通常是由鼠标或热键的动作引发的。事件驱动(Event Driver)就是鼠标或热键所引发的一系列动作。例如用户可以在 HTML 页面中移动鼠标到超链接上,而超链接的文字显示样式(颜色、字体等)会发生变化,这就是通过鼠标移动到指定对象上产生了一个事件,浏览器针对该事件进行了相应的处理所得到的结果。

4.5.2 事件处理方法

对事件进行处理的函数称为事件处理程序(Event Handler)或事件处理方法。由于客户端 JavaScript 通常是嵌套在 HTML 文件中,所以其事件处理一般会将 HTML 页面的具体元素和定义好的事件处理方法关联起来。

4.5.3 JavaScript 预定义的事件处理器

在 JavaScript 中给出了很多事件以及该事件的处理方法的定义,表 4-6 列出了其中常用的几种。

表 4-6 JavaScript 常用事件及处理方法

事　　件	说　　明	事件处理器
Click	当在页面对象上单击鼠标时产生的事件,如单击页面按钮 button、超链接 link 等对象	onClick
Load	从浏览器载入页面时产生的事件,通常是在<body>标记中发生	onLoad
Unload	当用户离开当前页面时产生的事件,通常其发生对象是<body>标记	onUnload
MouseOver	当鼠标移到指定对象上产生的事件,如超链接 link 对象	onMouseOver
MouseOut	当鼠标移开指定对象时产生的事件,如超链接 link 对象	onMouseOut
Submit	当用户提交表单内容时产生的事件	onSubmit

下面通过程序清单 4-5 所给出的程序来说明如何在 HTML 页面中使用 JavaScript 定义的事件处理器。

程序清单 4-5:

```
<!--Event.html 文件代码-->
<html>
<head>
```

```
<title>事件处理示例</title>
<script language="JavaScript">
function link(){
        if(window.confirm("确定链接到新浪主页？")==false)
            window.event.returnValue=false
    }
</script>
</head>
<body>
<br><br>
<center>
  <a href="http://www.sina.com.cn" onClick="link()">
    <h2>欢迎访问新浪主页！</h2>
  </a>
</center>
</body>
</html>
```

通过浏览器运行该程序可以看到如图 4-5 所示页面，当鼠标单击超链接时，会看到浏览器窗口弹出如图 4-6 所示对话框，若单击“确定”按钮则页面链接至新浪网主页，若单击“取消”按钮则页面停留在当前页。



图 4-5　程序 Event.html 运行结果(一)



图 4-6　程序 Event.html 运行结果(二)

4.6 JavaScript 内置对象

JavaScript 提供了一些常用的内置对象，使用户可以不用自己编写程序而是直接访问相应对象的属性和方法来实现某些操作。常用的内置对象除了在前面介绍的 Array 对象（见 4.2.6 小节）外，还有 String、Math 和 Date 对象。

1. String 对象

String 对象是针对字符串的操作定义的对象，创建 String 对象方法很简单，只要声明一个变量并将其初值赋值为一个字符串即创建了一个 String 对象的实例。如：

```
var s="student"     //创建了一个名为 s的 String对象
```

String 对象的常用属性和方法见表 4-7。

表 4-7 String 对象的常用属性和方法

	名　称	说　明
属性	length	统计字符串的字符个数，其结果为一个整数
方法	anchor(超链接名)	创建 HTML 的 anchor 标记
	big( )	以大写字体显示字符串
	small( )	以小写字体显示字符串
	italic( )	以斜体字体显示字符串
	bold( )	以粗体字体显示字符串
	blink( )	字符串闪烁显示
	fixed( )	固定字体高显示字符串
	fontsize(size)	控制字体以 size 值大小显示，size 取值为 1～7 的整数
	toLowerCase( )	将字符串的字符改为小写
	toUpperCase( )	将字符串的字符改为大写
	charAt(index)	返回位于字符串 index 下标位置的字符
	indexOf(str)	从字符串起始字符开始查找 str 字符串
	indexOf(str，start-position)	从字符串的 start-position 位置开始查找 str 字符串
	substing(position)	返回从字符串的 position 位置开始到结束字符的子串
	substing(position1，position2)	返回从字符串的 position1 位置开始到 position2 位置之间的子串

2. Math 对象

Math 对象是针对常数和数学运算定义的对象，包括三角函数、对数函数、指数函数等。Math 对象是一个静态对象，即它本身就是系统定义的一个实例，不需要再进行创建可直接使用，因此要调用 Math 对象定义的方法可以直接通过“Math. 方法名”等方式实现。其常用的属性和方法见表 4-8。

表 4-8　Math 对象的常用属性和方法

	名　　称	说　　明
属性	E	常数 e
	LN2	2 的自然对数
	LN10	10 的自然对数
	LOG2E	以 2 为底 e 的对数，即 $\log_2 e$
	LOG10E	以 10 为底 e 的对数，即 $\log_{10} e$
	PI	圆周率
	SQRT1_2	0.5 的平方根
	SQRT2	2 的平方根
方法	sin(val)	返回 val 的正弦值
	cos(val)	返回 val 的余弦值
	tan(val)	返回 val 的正切值
	asin(val)	返回 val 的反正弦值
	acos(val)	返回 val 的反余弦值
	atan(val)	返回 val 的反正切值
	power(bv, ev)	返回 bv 的 ev 次方
	sqrt(val)	返回 val 的平方根
	abs(val)	返回 val 的绝对值
	round(val)	返回 val 四舍五入的整数值
	max(val1,val2)	返回 val1 和 val2 值较大的一个
	min(val1,val2)	返回 val1 和 val2 值较小的一个
	random( )	返回 0～1 之间的随机数

3. Date 对象

Date 对象是针对日期和时间操作定义的对象，创建 Date 对象实例的语法形式如下：

```
var 对象实例名=new Date(参数)
```

其中参数可以没有，则系统会将当前日期和时间作为默认参数，如果要指定时间创建对象，则可以按照“月 日，年 时：分：秒”的形式或者将“年，月，日，时，分，秒”以整数形式设定参数。如：

```
var today=new Date( )
var date1=new Date("May 6,2008 10:21:30")
var date2=new Date(2008,5,6,10,21,30)
```

Date 对象没有定义属性，其常用的方法见表 4-9。

表 4-9　Date 对象的常用方法

	名　　称	说　　明
方法	getYear( )	返回年份
	getMonth( )	返回月份，其值范围为 0～11
	getDate( )	返回日数，其值范围为 1～31
	getDay( )	返回星期，其值范围为 0～6
	getHours( )	返回小时，其值范围为 0～23
	getMinutes( )	返回分钟，其值范围为 0～59
	getSeconds( )	返回秒，其值范围为 0～59
	getTime( )	返回表示时间的长整数，该值是从 1970 年 1 月 1 日起零点开始计算的秒数
	setYear(year)	设置年份，如果参数 year 是两位的数字，比如 setYear(91)，则该方法会理解为 1991。如果要规定 1990 年之前或 1999 年之后的年份，须使用四位数字，如 setYear(2008)
	setMonth(month)	设置月份，参数值范围为 0～11
	setDate(date)	设置日期数，参数值范围为 1～31
	setDay(day)	设置星期，参数值范围为 0～6
	setHours(hour)	设置小时，参数值范围为 0～23
	setMinutes(min)	设置分钟，参数值范围为 0～59
	setSeconds(sec)	设置秒，参数值范围为 0～59
	setTime(millisec)	设置长整数表示的时间，该值是从 1970 年 1 月 1 日起零点开始计算的秒数

小　　结

本章我们介绍了客户端 JavaScript 脚本语言的使用方法。主要包括以下几方面的内容：

① 脚本语言的基本概念；

② JavaScript 语言的书写方法、运行方式、基本数据类型和运算符；

③ JavaScript 语言的基本控制流程；

④ JavaScript 语言的函数定义和调用方法；

⑤ JavaScript 语言的事件处理和常用的内置对象。通过本章内容的学习，希望大家对脚本语言，特别是 JavaScript 有一个很好的了解。

练　习　4

1. 填空题

(1) 要将 JavaScript 代码嵌入 HTML 文件中，可以使用________标记标注要书写的

JavaScript 语句。

(2) 在 JavaScript 中可以使用关键字________进行变量的声明。

(3) 在 JavaScript 中没有定义基本的数组类型,我们可以通过它定义的内置对象________来创建数组对象。

(4) 在 JavaScript 中对象包含________和________两部分。对对象中属性的访问可以通过________的方式进行。

(5) 在 JavaScript 中函数的定义要使用关键字________。

(6) JavaScript 具有事件驱动的特点。对事件进行处理的函数称为________。

2. 选择题

(1) 下面对 JavaScript 描述错误的一项是________。

A. JavaScript 区分大小写。

B. 每条 JavaScript 语句可以用换行符或分号“;”作为结束标志。

C. 可以使用“//”或“/ * … * /”标记 JavaScript 的注释语句。

D. 在 JavaScript 中变量必须先用关键字“var”声明,然后才可以使用。

(2) 运行 JavaScript 语句:

```
var str="10"+"20";
document.write(str);
```

在浏览器上显示结果为________。

A. 10　　B. 20　　C. 30　　D. 1020

3. 实验题

使用 JavaScript 编写代码实现多幅图片每隔 2s 自动更换循环播放。

第 5 章　可扩展标记语言 XML

5.1　XML 基础

当今的社会是一个信息交换的社会。无论是个人还是团体还是社会机构，都已经利用 Web 技术发布关于各自领域的信息，进行信息交换和信息共享。网页无疑是利用 Web 发布信息的一种最重要和最常见的形式。

在前提下，在网络环境下的文件数据交换就成为一项非常重要的工作。可是，由于在计算机保存数据的文件格式大相径庭，利用文件进行数据交换成为一项技术挑战。当前文件由于应用领域的不同，使得数据在表示内容方面分成两种形式。

（1）结构性的数据。文件中的数据是经过分析和处理过的，具有一定的结构格式。例如，最常见的结构性的数据应用就是数据库。数据库中包含若干数据表，数据表中的数据按照二维表的形式保存。数据表和数据表之间可以体现一定的关系。比如学生记录表和课程表可以形成选课关系。结构性的数据形式方便信息处理、存储和应用。从这里可以发现这些结构性的数据有一个典型特征，即，多种数据组合形成信息，可以表达数据的含义，充分体现数据间相互关系。但是往往必须借助于特定软件工具表现数据。

（2）非结构性的数据。这是在计算机世界中大量存在的数据形式。例如，文本文件、电子邮件、html 网页、Word 2003 的 doc 文件等包含的数据。这种形式的数据与结构性数据比较，可以发现，它们可以表示一定数据内容，但是这些内容之间不能直接体现之间的联系，也没有有效的方式来表达这些数据之间是否存在关系。假设，用户需要将 Word 文件里的内容用数据库来表示，则存在数据内容难以区分的问题。

从上面的描述可以看到，结构化的数据具有良好定义数据结构，但是存在表示数据不足的缺陷；而非结构化的数据可以表现数据，但不能体现数据间的结构。如果，能将两者的优点结合起来，无疑是适应当前 Web 应用的趋势。如何将不同文件格式的文件数据，在保持数据一致性的同时，进行内容转换和良好地显示，达到实现数据的共享的目的，已经成为 Web 应用的一个重要课题。XML 在 20 世纪末期的出现为解决这一困境提出了有利的手段。

5.1.1　什么是 XML

XML 是 eXtenible Markup Language 的简称，它代表可扩展标记语言。要了解 XML，必须从 XML 语言的发展开始。XML 脱胎于国际标准化协会 ISO 定义的标准通用标记语言 SGML（Standard Generalized Markup Language）。SGML 语言是一个功能非常强大的标记语言，可以创建和扩展新的标记语言。但是 SGML 结构复杂，价格昂贵。这在一定程度上阻碍了它的广泛应用。与 SGML 对应的是 HTML。HTML 结构简单，善于表现数据，得到广大用户的支持。但是 HTML 在数据定义上出现混乱，用户往往无法正确理解

HTML 定义的数据。这就是结构化数据与非结构化数据都有的不足。为了解决数据内容定义和数据表现存在的问题，迫切需要一种新的技术。在这种情况下，1996 年，W3C 联盟组织 XML 工作组来研究新的标记语言 XML。1998 年 2 月，W3C 正式推荐了 XML 1.0 标准。

XML 语言的基本原理与 SGML 语言一致，就是只定义数据内容，不嵌入任何过程和处理内容。定义后的文件可以与多种技术结合重新编码形成多种的应用。XML 语言的重点就是定义数据的内容，然后结合其他技术表现数据。这就决定了 XML 便于数据存储、理解数据、数据交换以及跨平台应用。XML 克服了当时 SGML 语言结构复杂和 HTML 定义数据混乱的不足，充分体现将数据的内容和表现数据分离的思想。

从问世到现在，XML 已经成为 Web 应用的主流技术，具有广阔的应用前景。这是因为 XML 本身具有的特点。

(1) 扩展性。作为标记语言的一种，XML 也具有标记语言的特色：XML 是由标记(也称标签)构成的，用于定义数据。但是，XML 的标记不同于 HTML 和 XHTML 的预定标记，XML 标记必须是通过用户自定义的标记。由于用户可以定义特定领域的 XML 标记，生成新的标记语言。在第 6 章介绍的无线标记语言 WML(Wireless Markup Language)以及用于表示可扩展矢量图形 SVG(Scalable Vector Graphics)都是 XML 标记语言扩展出的一种新标记语言。这一特性也是 XML 之所以命名的主要原因。

(2) 灵活性。XML 与 HTML 明显不同。HTML 需要考虑文本、图像、用户图形界面等多种形式的语义的混合。这些多种形式的混合反而影响了 HTML 的进一步发展。形式间的微小变化，会引起 HTML 结构的复杂变动。所以可以解释从 1999 年 HTML 4.01 出现到 2007 年 HTML 5.0 的问世相距了 8 年之久。而 XML 只侧重数据的定义，它不考虑数据是如何处理以及数据的表现。对数据处理，以及数据的表现是通过与其他技术结合实现的。相比于 HTML，XML 形式更加灵活。XML 可以独立存在，也可以结合不同的环境表现不同的应用形式。

(3) 自描述性。XML 可用文件类型定义 DTD、XML Schema 描述数据。XML 文档是自我描述的。这一特性决定了 XML 文档便于理解。无论是计算机，还是开发人员，在理解 XML 文件定义的数据上不会存在困难。XML 自描述性的存在使得文档化的数据库成为可能。

(4) 简单性。XML 语言是简单的。简单性包含两层意思：一方面，用户学习使用 XML 非常简单。因为，如前述，XML 允许用户自定义标记。用户只要了解 XML 的语法，自行定义标记，学习使用起来非常简单方便。另一方面，XML 文件定义形式简单。XML 是 SGML 的子集，将 SGML 中非核心的、未被使用的和含义模糊的内容给予删除。从内容上说，XML 的篇幅约有 SGML 的 1/20。XML 继承了 SGML 的强大功能，却克服了 SGML 的复杂性。

XML 功能强大，使得 XML 一经推出，达到业内的广泛接受和使用。当前，XML 的主要应用于以下几个方面。

① XML 实现了数据和显示的分离：不同于 HTML，XML 是用于数据的描述。XML 文件可以与其他技术结合实现数据的表现。例如 XML 结合 CSS 层叠样式表，可以在浏览器上显示 XML 定义的数据出来。

② XML 实现数据的存储和共享：XML 文件实际上就是一个文本文件，不依赖于特定的系统。因此，XML 文件常常用做数据存储和共享。可以利用 XML 文件保存数据，根据需要将 XML 和数据库结合进行持久化存储。也可以将 XML 数据在不同平台上共享。

③ XML 实现数据的交换：当前计算机系统中文件格式不统一。XML 可以解决文件格式统一带来的问题。只要数据能转换成 XML 形式，就能实现在不同系统中交换数据。

④ XML 大量应用于 Web 应用。XML 为网络应用提供强大支持。不同的系统平台，可以通过 XML 实现数据的交换。特别是，SOAP（简单对象访问协议）和 XML-RPC（XML-远程过程调用协议）规范为 XML 实现分布式应用提供了可能。

5.1.2 XML 的相关技术

XML 文件本身只是描述数据的文本文件。脱离任何应用平台的 XML 文件是没有意义的。XML 往往和其他技术结合起来实现特定的应用。图 5-1 展示了与 XML 的相关技术。

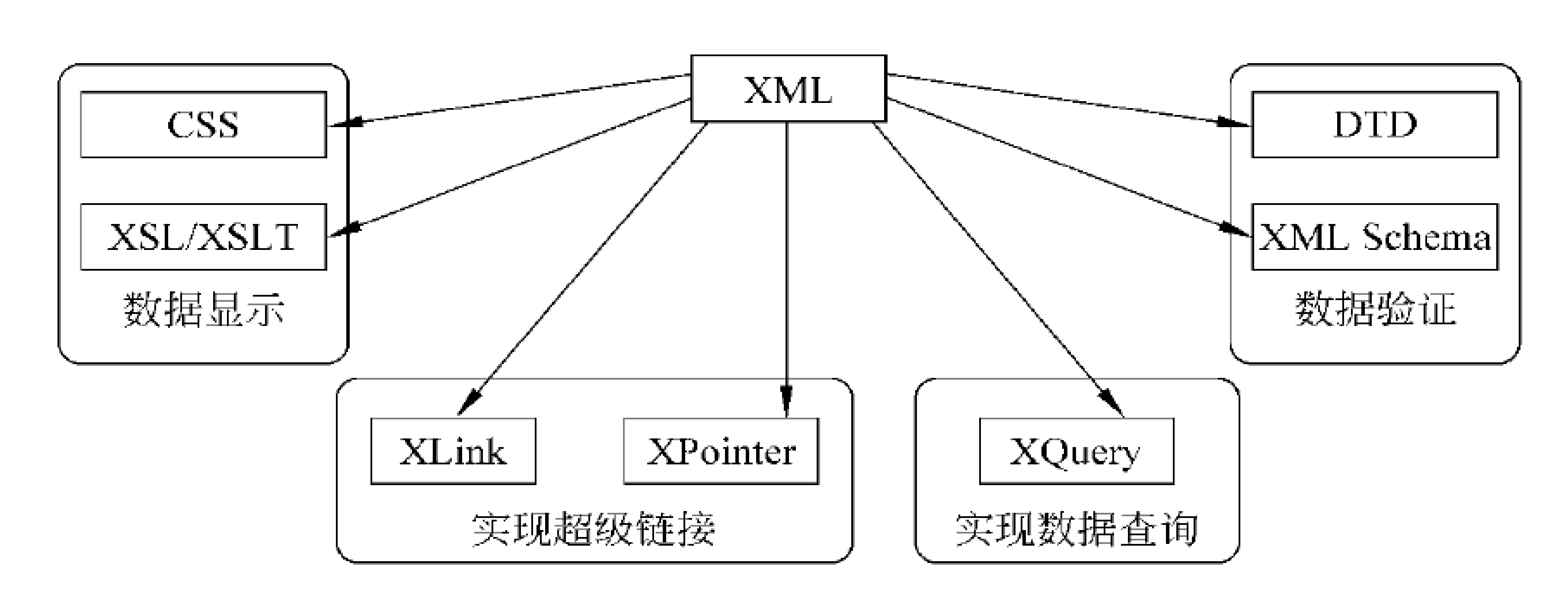


图 5-1 XML 的相关技术

1. CSS 层叠样式表

具体介绍见第 3 章。CSS 往往和 XML 结合，将 XML 文件中的数据以一定格式显示出来。

2. XSL/XSLT

XSL 是 eXtensible StyleSheet Language 的缩写，表示扩展样式表语言。XSL 由 3 个组成构成，它们是 XSLT（扩展样式表语言转换）、XPath 与 XSL-FO。其中 XSLT 用于将 XML 文件转换成其他格式，具体内容见 5.4 节。XPath 用于 XML 导航，是 XQuery 和 XPointer 的基础。而 XSL-FO 用于格式化 XML 数据。

3. XLink 和 XPointer

XLink 和 XPointer 结合使用。XLink 和 XPointer 可以实现访问链接资源的作用。

XLink 全称为 XML Linking Language，表示 XML 链接语言。XLink 可以插入 XML 元素，实现创建和描述资源的链接作用。

XPointer 是 XML Pointer Language 的简写，表示 XML 指针语言。XPointer 能让超级链接指向 XML 文档的片段。

4. XQuery

XQuery 是 XML 查询语言（XML Query Language）。XQuery 是一种查找和提取 XML 元素与属性的语言。XQuery 可以实现 XML 数据的查询的作用以及以 XML 形式出现的

数据。XQuery 常用于网络数据的提取。

5. DTD

DTD 是 Document Type Definition 文档类型定义。DTD 是定义 XML 文件的文档结构,可以在 XML 文件内部定义 DTD,或导入外部 DTD 至 XML 文件中。通过 DTD 可以实现 XML 文件数据的验证。具体内容见 5.2 节。

6. XML Schema

XML Schema 语言类似 DTD,可以定义 XML 文件的文档结构。通常把 XML Schema 规范视为 XML Schema Definition(XML 模式定义),简写成 XSD。XSD 可以实现 XML 文件的语法检查以及数据的验证。与 DTD 不同在于,XML Schema 是基于 XML 结构的。W3C 联盟认为 XML Schema 是 DTD 的后继者。具体内容见 5.2 节。

5.1.3 建立 XML 文件

XML 是一个自描述性的具有良构(Well-formed)的标记语言。可以用 XML 语言定义 XML 文件。在 W3C 推荐的 XML 1.0 说明书明确说明了"如果一个数据对象满足本规范中格式正确的定义时,它是一个 XML 文件"。为了对 XML 文件有一个直观的印象,请先看将电子邮箱定义成 XML 文件,代码见程序清单 5-1,执行结果如图 5-2 所示。

程序清单 5-1:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mailbox SYSTEM "DTD5-5.dtd">
<!--XML5-1.xml-->
<mailbox>
  <email>
        <from>chan@yahoo.com.cn</from>
        <to>liu@yahoo.com.cn</to>
        <subject>作业</subject>
        <date>Sun, 30 Dec 2007 22:01:25 +0800 (CST)</date>
        <body>请你看看我的实验报告有什么问题? 谢谢!
        </body>
        <attached-file>实验报告.doc</attached-file>
  </email>
  <email>
        <from>jiang@yahoo.com.cn</from>
        <to>liu@yahoo.com.cn</to>
        <subject>问好</subject>
          <date>Sun, 23 Jan 2007 22:01:25 +0800 (CST)</date>
          <body>节日快乐! 星期六聚会,请参加。</body>
  </email>
</mailbox>
```

XML 文件书写非常简单,大部分的文本编辑器都可实现文件编辑。所有的 XML 文件必须保存为文件扩展名为.xml 的文件,如程序清单 5-1 的 mail.xml。XML 文件是由数字字符和各种类型的标记所构成,从逻辑上可以将 XML 文件主要由:处理指令、文件声明、标

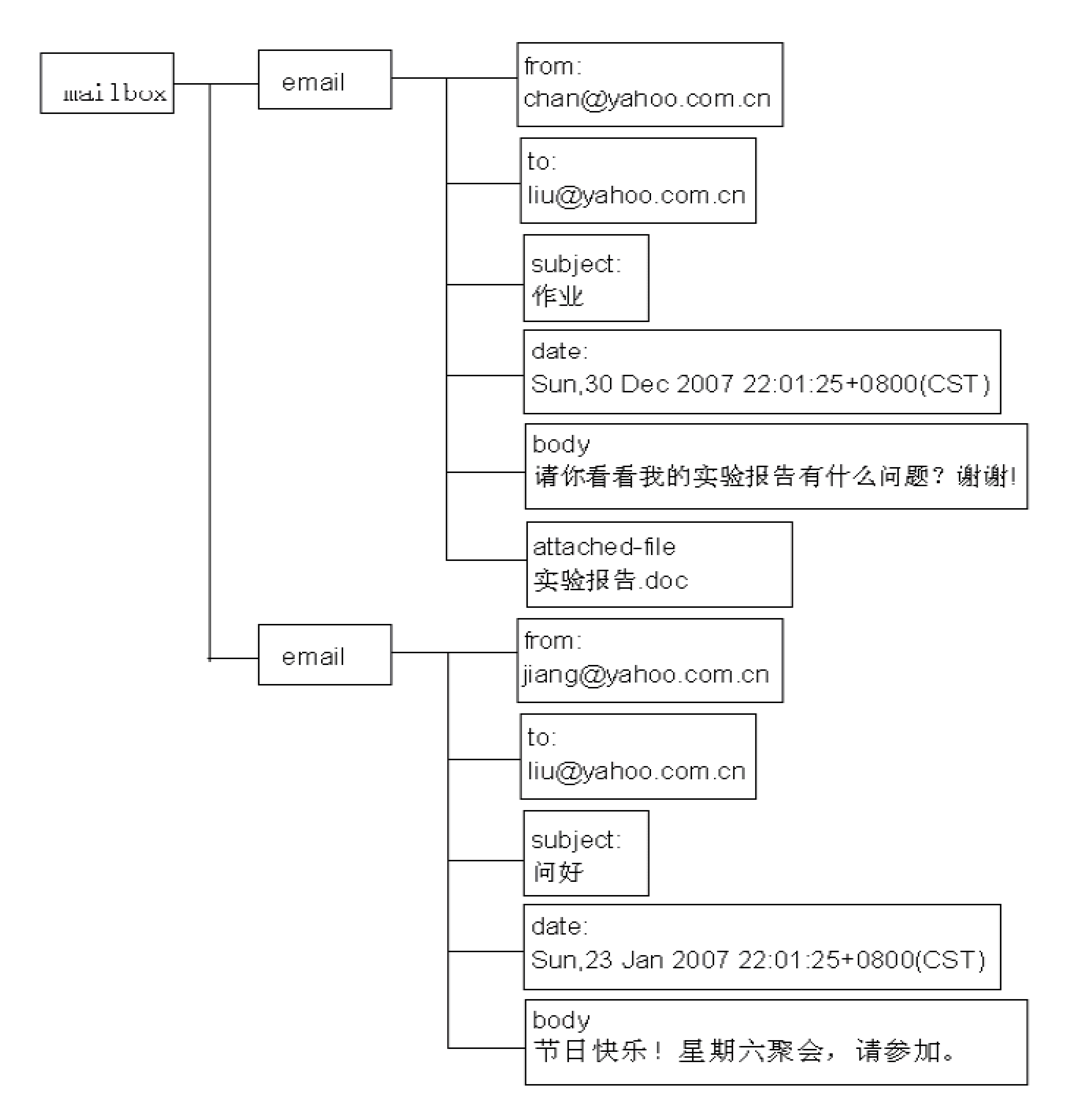


图 5-2　程序清单 5-1 对应的 XML 树

记、实体引用、注释、CDATA 片段组成。

1. 处理指令

XML 文件的处理指令是向应用程序传递的特殊指令。处理指令必须处于 XML 文件首部。在程序清单 5-1 的 mail.xml 文件第一行就是一个处理指令“＜？ xml version＝"1.0" encoding＝"UTF-8"？ ＞”。这个处理指令向应用该 XML 文件的程序说明该 XML 文件的版本是 1.0，采用的字符集是“UTF-8”。常见支持中文的字符集见表 5-1。这一处理指令还为未来的 XML 向下兼容提供了保证。

表 5-1　常见支持中文的字符集

字符集	说　明	字符集	说　明
GB2312	支持简体中文	UTF-8	支持世界上大部分语言
UTF-16	支持世界上大部分语言	BIG5	支持繁体中文

注意：用 Windows 操作系统中自带的“记事本”编写 XML 文件，如果字符集是 UTF-16 或 UTF-8，请保存文件时将“编码”项写成对应的字符编码，而不是默认的 ASCII 码。

2. 文档类型定义（DTD）

XML 文件通过 DOCTYPE 文件类型声明说明文件的使用的文法，定义了 XML 文件的元素、属性，以及实体等多项内容，具体指明了 XML 文件结构。XML 文件确定的使用文法，有两种方式：

（1）指明外部的 DTD 文件。

（2）可以通过在 XML 的内部的 DTD。文件类型定义具体的位于文件首部，第一个标记元素之前。

程序清单 5-1 第 2 行“<!DOCTYPE mailbox SYSTEM "DTD5-5.dtd">”就是指明了外部 DTD5-5.dtd 作为 XML5-1.xml 文件的文法说明，并且指出了 XML5-1.xml 的根元素是 mailbox 标记。关于 XML 文件的 DTD 具体内容见 5.2.1 小节。

3. 标记

XML 文件中最主要的内容是由大量的元素构成的。XML 文件元素可以是由成对标记中封闭内容构成，也可以是不包含内容的空元素。这和 HTML 元素的概念近似。XML 文件中的标记具有几点语法要求，以符合 XML 文件的良构性。

（1）XML 的标记区分大小写；标记<applet>和<Applet>在 XML 文件中就是代表两种不同的标记。

（2）XML 的标记必须是封闭的。这意味着，XML 的非空元素必须有起始标记形如“<标记名>”开始一个元素，用结束标记形如“</标记名>”来结束元素。对于 XML 的空元素也必须增加空格和“/”，形如“<标记名 />”实现空元素的封闭。

（3）XML 文件中必须有一个根元素。所有其他元素必须处于根元素的内部，成为根元素的子元素。XML 文件中的元素可以嵌套其他元素，其中嵌套其他元素的元素为父元素，而嵌套在内部的元素为子元素。所以在 XML 文件中必须正确嵌套，不能重叠。XML 文件的这一特点可以很好地定义具有树型结构的数据。程序清单 5-1 定义电子邮箱的树状结构，如图 5-2 所示。

（4）在 XML 文件中标记的属性必须通过双引号包含起来，不能忽略。这与 XML 文件的其他部分的要求一致。

4. 实体引用

实体引用实际上就是引用具有特定意义的字符对象。引用实体有两种形式。

（1）“& 字符串；”。

（2）“&# 实体编号”，其中实体编号是由十六进制的数字构成。

5. 注释

XML 文件也定义注释，注释的形式同 HTML 注释一致，形如“<!-- 注释 -->”。注释为 XML 文件提供必要的解释说明。在应用程序使用 XML 文件时，XML 文件的注释会被忽略。

6. PCDATA 段和 CDATA 段

在 XML 元素中，标记之间包含的内容就是 PCDATA 段。在程序清单 5-1 所示，from 元素中包含的文本内容“chan@yahoo.com.cn”就是一个 PCDATA。对于 PCDATA 块在 XML 文件会被应用程序解析。与 PCDATA 对应的是 CDATA 段。

CDATA 段具有特定的格式标明它的特殊性。CDATA 片段不会被应用程序进行解析，而是全部显示。在 CDATA 段中的标记与实体和其他内容一起会原样显示。CDATA 片段定义形如：

```
<![CDATA[
    文本...
]]>
```

CDATA 段的示例代码见程序清单 5-2，function getNumber()的内容并没有作为元素进行解析处理，只被浏览器作为文本信息全部显示。由于单独使用<、& 符号会造成 XML 文件解析错误。所以，在大量使用的<、& 符号可以考虑使用 CDATA 段。程序清单 5-2 执行结果如图 5-3 所示。

程序清单 5-2：

```
<?xml version="1.0" encoding="GB2312"?>
<!--XML5-2.xml-->
<book>
   <bookname>计算机应用</bookname>
   <author>程地</author>
   <publisher>××出版社</publisher>
   <script>
   <![CDATA[
         function getNumber(number){
         if(number<0&&number>MAXLEN)
                  return
         else{
                  number++;
                  return number;
                  }
         }
   ]]>
   </script>
</book>
```

```
<?xml version="1.0" encoding="GB2312" ?>
- <book>
    <bookname>计算机应用</bookname>
    <author>程地</author>
    <publisher>XX出版社</publisher>
  - <script>
    - <![CDATA[
              function getNumber(number){
              if(number<0&&number>MAXLEN)
                    return
              else{
                    number++;
                    return number;
                    }
              }
      ]]>
    </script>
  </book>
```

图 5-3　CDATA 块在浏览器运行示例

注意：在 CDATA 片段中不能在文本信息中包括“]]>”，以避免造成 CDATA 片段错误结束的标记。

5.1.4　XML 的命名空间

XML 语言允许一个 XML 文件可以引用其他 XML 文件。这就存在一个问题，如果多个 XML 文件定义了名称相同的元素，会产生命名冲突的问题。为此，XML 语言中通过命名空间来解决问题。

XML 命名空间是一组关于元素和属性命名唯一的集合的名称。通过 XML 命名空间可以标识和区分不同的元素和属性。W3C 于 1999 年年初次推荐使用 XML 命名空间规范。在该规范中指出了如下 XML 命名空间的定义形式。

```
<命名空间前缀:元素名   xmlns:命名空间前缀="命名空间 URI">
```

按上述形式定义 XML 命名空间后，可以直接使用命名空间前缀来表示命名空间标识符。为了了解 XML 命名空间，可以观察如下代码：

```
⋮
<sl:studentlist xmlns:sl="http://www.xschool.com/">
<sl:student>
    <sl:name>张珊</sl:name>
    <sl:gender>女</sl:gender>
```

```
        <school:name xmlns:school="http://www.schoollist.com">
          ×××市北京路 232 号
        </school:name>
    </sl:student>
    </sl:studentlist>
    ⋮
```

可以发现，作为 studentlist 元素的子元素每一个都标记命名空间前缀。这样的表示方式比较烦琐。可以采用默认的命名空间形如“xmlns=命名空间 URI”以简化文件。将上述的代码可以改写成如下内容。

```
    ⋮
    <studentlist xmlns:sl="http://www.xschool.com/"
    xmlns:school="http://www.schoollist.com" >
        <student>
            <name>张珊</name>
            <gender>女</gender>
            <school:name >×××市北京路 232 号 </school:name>
        </student>
    </studentlist>
    ⋮
```

其中作为 studentlist 子元素的 student、name、gender 是命名空间为“http://www.xschool.com”。而子元素 school:name 为了避免和 name 子元素发生冲突，仍引用命名空间前缀 school。

5.1.5 XML 的数据岛(XML Data Inland)

XML 数据岛指能被微软的 Internet Explorer 5.x 以上版本识别，以及嵌入到 HTML 中的 XML 数据。使用 XML 数据岛非常方便，通过 XML 数据岛可以将 XML 数据作为一个整体直接传入到具有 HTML 的网页中，不需要脚本语言就可以实现。要定义 XML 数据岛，这需要在 HTML 代码按照下列形式将一个 XML 文件声明 XML 数据岛。

```
<xml id="数据岛名" src="XML 文件 URL">
```

要使用数据岛还需要实现数据绑定，即，需要将 XML 数据绑定到具体 HTML 元素中。在 HTML 的元素如 table 通过属性“datasrc”指明使用具体名称的 XML 数据岛。然后在 HTML 的元素如 div、span 中用属性“datafld”来绑定到具体的 XML 数据。下列程序清单 5-3 的 HTML5-3.html 展示了将 XML5-1.xml 作为数据岛在 HTML 中显示，执行结果如图 5-4 所示。

程序清单 5-3：

```
<!--HTML5-3.html-->
<html>
<head><title>应用数据岛</title></head>
<body>
<xml id="emaildata" src="XML5-1.xml"></xml>
```

```
<table border="1" datasrc="# emaildata">
  <tr>
    <td><span datafld="from"></span></td>
    <td><span datafld="to"></span></td>
    <td><span datafld="subject"></span></td>
    <td><span datafld="body"></span></td>
  </tr>
</table>
</body>
```

chan@yahoo.com.cn	liu@yahoo.com.cn	作业	请你看看我的实验报告有什么问题？谢谢！
jiang@yahoo.com.cn	liu@yahoo.com.cn	问好	节日快乐！星期六聚会，请参加。

图 5-4　XML 数据岛的应用

注意：XML 数据岛只能在 Internet Explorer 5.x 以上浏览器使用，其他浏览器并不支持。在使用 XML 数据岛时要谨慎。

5.2　XML 验证机制

XML 文件必须符合 XML 规范。在具体表现上，XML 文件不单具有严格的 XML 语法要求，还要求 XML 文件定义结构在文档中必须保持一致。为了保证一个 XML 文件的良构（Well-formed）特点和确保 XML 文件符合定义规则，有必要对 XML 文件进行验证。

实际上，XML 验证就是按照用模式语言如 XML Schema 或 DTD 或其他模式语言定义的文件结构，对 XML 文件内容强制执行规则的过程。具体的验证可以通过特定的验证工具来实现。当前常用的验证工具主要有嵌入在 IE 浏览器的 MSXML 解析器。也可以通过一些网站实现 XML 文件的验证，如 W3C 联盟网站提供的“Validator for XML Schema”。从本质上说，XML 验证机制就是让 XML 解析器按照 XML Schema 或 DTD 确定 XML 文件的规则来理解 XML 文件结构、文件的标记元素、属性等内容。

DTD 和 XML Schema 常用于 XML 验证机制。从语法的角度来说，两者不同之处在于 DTD 具有不同于 XML 的语法规则，而 XML Schema 是基于 XML，和 XML 具有一致的语法要求。在本节中将对它们详细介绍，具体了解两者的不同。

5.2.1　文档类型定义 DTD

DTD（Document Type Definition，文档类型定义）是一种 XML 验证机制，是 XML 标准的一部分。DTD 可以确定 XML 文件的合法组成，定义 XML 文件的结构。DTD 可以定义在 XML 文件内部，称为内部 DTD。XML 文件也可以通过语句实现外部 DTD 的引用。无论在内部 DTD 还是外部 DTD，都需要指定 XML 文件模块中的元素、属性和实体。在下面将对这些内容的实现进行说明。

1. 内部 DTD

XML 文件中通过 DOCTYPE 实现内部 DTD 的定义。具体 DTD 内容包括在

DOCTYPE 中。形如：

```
<!DOCTYPE 根元素名[
  <!-- 定义元素 -->
 ⋮
]>
```

下面的程序清单 5-4 就是内部 DTD 的一个应用。

程序清单 5-4：

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE mailbox[
<!ELEMENT mailbox ((email+))>
<!ELEMENT email ((from, to, subject, date, body, attached-file))>
<!ELEMENT to (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT attached-file (#PCDATA)>
]>
<!-- XML5-4.xml-->
<mailbox>
  <email>
        <from>jiang@yahoo.com.cn</from>
        <to>liu@yahoo.com.cn</to>
        <subject>问好</subject>
      <date>Sun, 23 Jan 2007 22:01:25 +0800 (CST)</date>
        <body>节日快乐！ 星期六聚会，请参加。
      </body>
  </email>
</mailbox>
```

2. 外部 DTD

外部 DTD 是导入到 XML 的 DTD。首先，用户需要按 DTD 语法规则定义一个文件扩展名为"dtd"的文件。在这个 DTD 文件中确定 XML 文件的结构及组成。然后在 XML 文件中通过 DOCTYPE 导入该文件。具体的形如："<!DOCTYPE 根元素 SYSTEM "文件名">"。在程序清单 5-1 中就展示了外部文档声明的实例。其中，外部 DTD 文件"DTD5-5.dtd"的内容见程序清单 5-5。

程序清单 5-5：

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD5-5.dtd-->
<!ELEMENT mailbox (email+)>
<!ELEMENT email (from, to, subject, date, body, attached-file?) >
<!ELEMENT to (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT from (#PCDATA)>
```

```
< !ELEMENT date (#PCDATA)>
< !ELEMENT body (#PCDATA)>
< !ELEMENT attached-file (#PCDATA)>
```

3．DTD 元素

在 DTD 中声明 XML 文件中元素的元素是 DTD 元素。DTD 元素可以说明 XML 的元素标记、特定元素的子元素以及元素包含的内容。DTD 元素可以定义 XML 文件的元素结构，以及反映元素间的相互关系。DTD 元素通过＜!ELEMENT...＞实现定义，语法形式如下：

```
< !ELEMENT 元素名称 内容>
```

（1）定义一个空元素。XML 的空元素在 DTD 定义形如“＜!ELEMENT 元素名称 EMPTY＞”。例如，XHTML 的＜hr /＞用 DTD 定义为：

```
< !ELEMENT hr EMPTY>
```

（2）定义包含内容的元素。XML 包含内容的元素具有 3 种形式。

① ＜!ELEMENT 元素名称（＃PCDATA）＞。＃PCDATA 表示元素的内容是字符数据。例如，可以将 XHTML1.0 的 b 粗体元素（形如＜b＞hello＜/b＞）用 DTD 定义为：

```
< !ELEMENT b (#PCDATA)>
```

② ＜!ELEMENT 元素名称（＃CDATA）＞：＃CDATA 表示元素内容可以包含了解析器中不可以解析的字符。

③ ＜!ELEMENT 元素名称 （ANY）＞：ANY 表示元素内容可以是任何可以被解析器理解的数据组合。例，＜!ELEMENT email（ANY）＞

（3）定义包含子元素的元素。XML 文件体现了树状结构，元素可以嵌套，即一个元素中包含了多个子元素。为了与实际情况相符，XML 文件要求反映出子元素可以选择的状况。例如，在留言板中留言和正文反映同一性质，不能同时出现，在实际应用中可以选择。为此，DTD 元素可以定义子元素序列。子元素序列的定义形式如下：

① ＜!ELEMENT 元素名称（子元素名称）＞：表示元素中包含一个子元素。

② ＜!ELEMENT 元素名称（子元素 1 名称，子元素 2 名称，…）＞：表示一个元素包含了一个子元素序列。

③ ＜!ELEMENT 元素名称（子元素 1 名称|子元素 2 名称|…）＞：表示一个元素可以包含多种子元素，这些子元素不能同时嵌套在元素中。

有时，需要对元素出现的次数进行限制。DTD 元素定义中可以结合一些具有特定含义的符号来实现，具体内容见表 5-2。

表 5-2　规定 DTD 元素出现次数的符号

字　符	说　　明	示　　例
＋	元素最少出现一次	＜!ELEMENT mailbox（email＋）＞
＊	元素可以出现零次或多次	＜!ELEMENT email（from，to＊，body）＞
?	元素可以出现零次或一次	＜!ELEMENT email（from，to，body?）＞

(4) 定义混合型的元素。混合型的元素指一个元素中包含的内容具有多种形式，元素内容可以是子元素也可以是字符数据。观察例子"<!ELEMENT email (#PCDATA|from|to|body)*>"，可以发现 email 元素可以包含字符数据以及子元素 from、to 和 body，这些子元素可以出现零次或多次。程序清单 5-6 和程序清单 5-7 分别展示了 DTD 定义的 DTD5-6.dtd 以及对应的 XML 文件 XML5-7.xml 示例。

程序清单 5-6：

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD5-6.dtd-->
<!ELEMENT mailbox (#PCDATA|email)* >
<!ELEMENT email (#PCDATA|from|to|subject|body)* >
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

程序清单 5-7：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mailbox SYSTEM "DTD5-6.dtd">
<!--XML5-7.xml-->
<mailbox>作业邮件
  <email>
      <from>chan@yahoo.com.cn</from>
      <to>liu@yahoo.com.cn</to>
      <subject>问题</subject>
      <body>请你看看我的实验报告有什么问题？</body>
  </email>
  <email>
      <from>jiang@yahoo.com.cn</from>
      <to>liu@yahoo.com.cn</to>
      <to>wang@yahoo.com.cn</to>
      <subject>问好</subject>
      <body>节日快乐！ 星期六聚会，请参加。江五    </body>
  </email>
</mailbox>
```

4. DTD 属性

DTD 属性是确定 XML 元素的属性的规则。DTD 属性的声明是通过<!ATTLIST…>来实现的。具体的语法要求如下：

```
<!ATTLIST 元素名称 属性名称 属性类型  默认值
          …>
```

定义属性，必须指定属性的归属：元素名称，以及属性的性质，即确定属性的名称、属性的类型、属性的默认值。有几点说明如下。

(1) 元素名称：是说明要定义属性的元素。

（2）属性名称：确定属性的名字。
（3）属性类型：指定属性的类别，具体内容见表 5-3。

表 5-3　DTD 属性类型

类　型	说　明
CDATA	字符数据
(enum1|enum2|…)	枚举数据
ID	唯一的编号，ID 对应的属性值必须由字母开头
IDREF	其他元素的编号
IDREFS	其他元素的编号列表
ENTITY	实体属性
ENTITYS	实体列表，实体间用空格分开
NMTOKEN	定义合法的 XML 名称
NMTOKENS	定义 XML 名称的列表
NOTATION	定义符号名称的属性
xml:	预定义属性

（4）默认值：确定给属性值在没有指定时自动具有的内容，具体内容见表 5-4。

表 5-4　DTD 属性默认值类型

默认值类型	说　明
值	属性的默认值
#REQUIRED	属性值是必不可少的
#IMPLIED	属性值不是必需的，是可选的。一般，属性不具有默认值，采用 #IMPLIED
#FIXED 值	属性值是固定给定的值

将上述程序清单 5-6 的 DTD5-6.dtd 改写成程序清单 5-8 的 DTD5-8.dtd。其中，为 email 增加 id 属性（注明邮件的编号，编码唯一，内容必需）和 format 属性（注明使用的邮件格式，取固定值"Simple"）。from 和 to 元素各添加一个 catalog 属性（注明邮件地址的类别，值为枚举类型），from 元素的 catalog 属性具有默认值"friend"，而 to 元素的 catalog 属性是可选属性。对应的 XML 文件示例见程序清单 5-9 的 XML5-9.xml。

程序清单 5-8：

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD5-8.dtd-->
<!ELEMENT mailbox (#PCDATA|email)*>
<!ELEMENT email (from, to*, subject,(body?|message?))>
<!ATTLIST email
        id ID #REQUIRED
        format CDATA #FIXED "Simple"
```

```
>
<!ELEMENT from (#PCDATA)>
<!ATTLIST from catalog (friend|company|family|other) "friend">
<!ELEMENT to (#PCDATA)>
<!ATTLIST to catalog (friend|company|family|other) #IMPLIED>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT message (#PCDATA)>
```

程序清单 5-9：

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XML5-9.xml-->
<!DOCTYPE mailbox SYSTEM "DTD5-8.dtd">
<mailbox>
<email id="liu1" format="Simple">
    <from>liu@yahoo.com.cn</from>
    <to>wang@yahoo.com.cn</to>
    <subject>问好</subject>
    <body>你好,有空和我联系</body>
</email>
<email id="liu2">
    <from>wang@yahoo.com.cn</from>
    <to catalog="family">wang_he@yahoo.com.cn</to>
    <subject>问好</subject>
    <message>假期回家</message>
</email>
<email id="liu3">
    <from>wang@yahoo.com.cn</from>
    <to catalog="company">he_jiang@yahoo.com.cn</to>
    <subject>请假批准</subject>
    <message>春假批准,时间是 2月 1日至 2月 20日。</message>
</email>
</mailbox>
```

5. DTD 实体

实体实质上是一个变量,可以快速地引入普通文本或特殊含义的字符。DTD 实体用于定义 XML 的实体,是 XML 实体引用实现的保证。DTD 的实体有两种形式:内部实体和外部实体。

(1) 内部实体。内部实体是在 DTD 文件内部中声明的实体。定义的语法形如:"<!ENTITY 实体名 实体值>"。例如在 XHTML 空格实体( ),用 DTD 定义为"<!ENTITY nbsp " ">",浏览器会解析成空格。

(2) 外部实体。外部实体是引入其他文件的实体。声明的语法形式如:"<!ENTITY 实体名 SYSTEM "URI/URL">"。其中,"URI/URL"指定引入实体的 URL 地址。

5.2.2　XML 模式定义语言(XML Schema Definition Language)

在了解 XML Schema 之前,请观察下列程序清单 5-10,studentlist.dtd:

程序清单 5-10:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT studentlist (student+)>
<!ELEMENT student (id,name,birthday,gender)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthday (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
```

在上述 DTD 代码中,name、birthday、gender 分别表示姓名、出生日期及性别。实际上 name 的数据类型与出生日期是不同的。但是,这些实际情况在 DTD 文件只能用 #PCDATA 表示,不能体现两者的不同。另外,在属性表达上,DTD 虽然能表达枚举类型,但不能表示一个具体取值范围。另外,DTD 语法复杂,学习困难。DTD 的这些问题迫切需要解决。在此前提下,XML Schema 应运而生。

XML Schema 定义语言(XSDL)与 2001 年 5 月 2 日成为 W3C 推荐标准的一员。它也是 XML 验证机制的一种,可以表达以及验证 XML 文件结构。与 DTD 相比,XML Schema 定义语言在形式上最明显的特点就是基于 XML,具有 XML 语言的同样的语法要求,继承了 XML 语言的良构性、灵活性、扩展性和简单性。此外,XML Schema 定义语言克服了 DTD 中表达元素属性和元素内容的约束性不足的特点。利用 XML Schema 定义语言定义的文件称为 XML Schema 文件,文件的可扩展名为".xsd"。接下来,将从 XML Schema 文件的基本结构、数据类型、元素声明、属性声明、命名空间来了解 XML Schema 技术。

1. XML Schema 文件的基本结构

XML Schema 文件实际上就是一个 XML 文件,不同在于指定了这个 XML 文件的根元素必须是 schema,表示 XML 文件的模式结构定义。形式如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">
   ⋮
</xs:schema>
```

在上述形式中,"xs:"(注意有时也可以写成"xsd:")表示文件中的元素和数据类型来自于 XML Schema 的命名空间,空间位于"http://www.w3.org/2001/XMLSchema"中。"elementFormDefault"表示 schema 的元素属于目标命名空间的默认写法。如果取值为"qualified",表示任何使用该 XSD 文件的元素数据的 XML 文件,必须使用同一命名空间,否则取值为"unqualified"。XSDL 定义了不同性质的元素,可以定义 XML 文件的元素、属性以及包含的内容。详细内容见表 5-5。

表 5-5　XML Schema Definition Language 的常见元素

元　　素	说　　明
schema	定义 schema 根元素
element	定义 XML 元素
attribute	定义 XML 元素的属性
simpleType	用户自定义简单类型，声明 XML 元素的文本内容和属性
complexType	用户自定义复杂类型，以及声明 XML 元素的子元素
restriction	定义 XML 元素内容的约束条件

值得注意的是，为了运用 xsd 文件进行验证，xml 文件内必须对此进行说明，声明形式如下：

```
<根元素 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="xsd文件">
```

用 XML Schema 重新定义程序清单 5-10 中类似树状结构。具体内容见程序清单 5-11。

程序清单 5-11：

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XSD5-11.xsd-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="studentlist">                  <!-- studentlist定义 -->
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="student" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="student">                      <!-- student定义-->
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="id"/>               <!-- 引用子元素 id-->
                <xs:element ref="name"/>             <!-- 引用子元素 name-->
                <xs:element ref="birthday"/>         <!-- 引用子元素 birthday-->
                <xs:element ref="gender"/>           <!-- 引用子元素 gender-->
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="id">                           <!-- id定义-->
        <xs:simpleType>
            <xs:restriction base="xs:string"/>       <!-- id元素的类型限定为字符串-->
        </xs:simpleType>
```

```
        </xs:element>
        <xs:element name="name">                    <!-- name 定义 -->
            <xs:simpleType>
                <xs:restriction base="xs:string"/>  <!--name 元素的类型限定为字符串-->
            </xs:simpleType>
        </xs:element>
        <xs:element name="birthday">                <!-- birthday 定义 -->
            <xs:simpleType>
                <xs:restriction base="xs:date"/>    <!-- birthday 元素的类型限定为日期-->
            </xs:simpleType>
        </xs:element>
        <xs:element name="gender">                  <!-- gender 定义 -->
            <xs:simpleType>
                <xs:restriction base="xs:string"/>
            </xs:simpleType>
        </xs:element>
</xs:schema>
```

对应的 XML 文件见程序清单 5-12 的 XML5-12.xml。

程序清单 5-12：

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XML5-12.xml-->
<studentlist xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XSD5-11.xsd"><!-- 说明绑定 studentlist.xsd -->
    <student>
        <id>90123</id>
        <name>王一</name>
        <birthday>1990-12-10</birthday>
        <gender>男</gender>
    </student>
    <student>
        <id>90124</id>
        <name>刘萨</name>
        <birthday>1992-02-19</birthday>
        <gender>男</gender>
    </student>
</studentlist>
```

2. XML Schema 元素的声明

XML Schema 元素是用“<xs:element>”来实现 XML Schema 元素的定义，常见的语法格式如下：

```
<xs:element name="元素名" type="数据类型" default="默认值" maxOccurs="最大取值" minOccurs="最小取值" fixed="固定取值" ref="引用元素名">
```

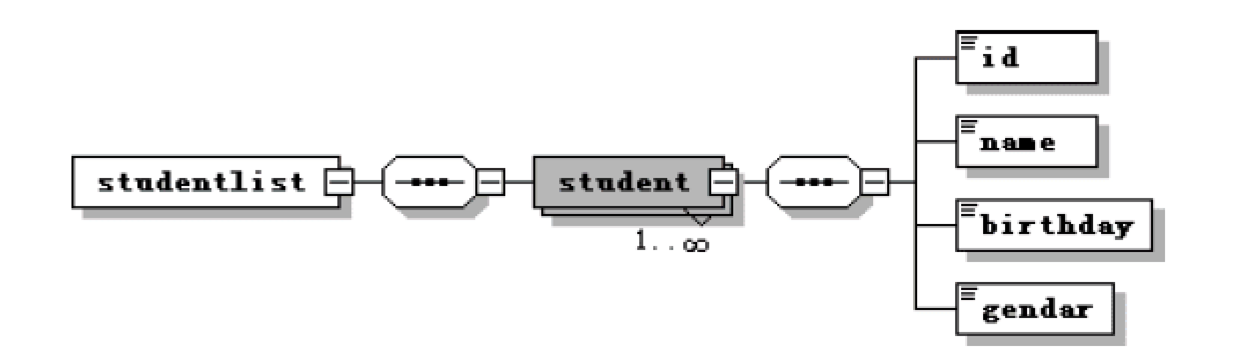


图 5-5　studentlist 的模式结构

说明：

（1）name 表示定义 XML Schema 元素的命名；该属性是 XML Schema 元素定义必不可少的属性。

（2）type 表示 XML Schema 元素的取值的性质，指明属于的数据类型，可以是 XML Schema 内置基本数据类型（具体内容见表 5-6），也可以是用户自定义数据类型。

表 5-6　XML Schema Definition Language 的基本数据类型

数据类型	说　　明	数据类型	说　　明
xs:string	字符串	xs:gYearMonth	按 Gregorian 历的年月
xs:boolean	布尔数	xs:gYear	按 Gregorian 历的年
xs:decimal	十进制数	xs:gMonthDay	按 Gregorian 历的月日
xs: precisionDecimal	精确十进制数，如－0	xs:gDay	按 Gregorian 历的日
xs:float	单精度实数	xs:gMonth	按 Gregorian 历的月
xs:double	双精度实数	xs:hexBinary	十六进制编码的二进制数
xs:duration	持续时间的长度	xs:base64Binary	六十四进制编码的二进制数
xs:dateTime	指定日期时间	xs:QName	XML 限制名
xs:date	日期	xs:anyURI	URL
xs:time	时间，格式 hh:mm:ss:ss	xs:NOTATION	标记

（3）default 表示 XML Schema 元素自动带有的默认值，除非改变。

（4）maxOccurs 和 minOccurs 分别表示 XML Schema 元素取值最大和最小范围，只能取非负整数。如果 maxOccurs 取值为“unbounded”，表示不受限制。如果 XML Schema 元素没有定义 maxOccurs 和 minOccurs 属性，这两个属性的值默认为 1。

（5）fixed 表示元素取一个固定的值，不能改变。

（6）ref 表示一个元素的引用，通常表示引用元素的子元素。该属性往往结合 XML Schema 复杂元素的定义。

从程序清单 5-11 中可以看出，XML Schema 元素具有两种形式，一种是简单元素，即，XML Schema 元素只有元素文本内容，不包括子元素和属性，如 id 元素等。除此之外为复杂元素。

3. XML Schema 的属性声明

XML Schema 元素的属性是通过“<xs:attribute>”来定义的。通过属性的定义可以更好地描绘元素的特征和性质。XML Schema 属性声明的语法形式如下：

```
<xs:attribute name="属性名称" type="数据类型" default="默认值" fixed="固定值" id="标号" use="使用性质">
```

(1) name 是一个必选项，表示属性的名称。

(2) type 表示属性所属的数据类型，可以是 XML Schema 内置的数据类型(见表 5-6)，也可以是用户自定义的数据类型；

(3) default 与 fixed 分别表示属性的取值为默认值与固定值，两者不能同时出现。

(4) id 表示属性的唯一标号，取值只能是由字符开头。

(5) use 表示属性的性质，可以表示属性是“optional”(可选的)、“required”(必选的)以及“prohibited”(禁止的)，默认值为 optional 表示可选的。

例如语句“<xs:attribute name="type" type="xs:string">”表示定义一个类型为字符串的 type 属性。

4. 简单元素

简单元素只定义元素的内容。对于元素的内容可以根据实际情况，带有默认值或固定值。例如 XSDL 描述以下语句：

```
<xs:element name="class" type="xs:string" fixed="计算机 2 班">
```

该语句对应的 XML 内容可以是“<class>计算机 2 班</class>”。

5. 复杂元素

不包括简单元素的元素就是复杂元素。复杂元素表示多种特殊情况的元素，复杂元素可以是嵌套子元素(如 studentlist 元素)、嵌套子元素和包含文本的元素、空元素、包含属性的元素、包含属性和文本内容的元素。接下来对这些复杂元素的定义做一个介绍。

(1) 嵌套子元素的元素。嵌套子元素的元素的定义基本形式如下：

```
<xs:element name="元素">
  <xs:complexType>
    ⋮
  </xs:complexType>
<xs:element>
```

在程序清单 5-11 中就展示了这类元素的定义。子元素并不能在 xs:complexType 元素内直接定义，必须通过一些 XML Schema 指令功能的元素来说明子元素的顺序和次数。这些 XML Schema 元素具体内容见表 5-7。

表 5-7 XML Schema 指令功能子元素

元　　素	说　　明
xs:all	表示 XML 子元素可以以任意顺序排列
xs:sequence	表示 XML 子元素必须按照规定的顺序排列
xs:choice	表示 XML 子元素为可选项，可以出现，也可以不出现
xs:group	表示一组 XML 元素
xs:simpleContent	不定义元素，但定义元素内的内容、属性或简单类型
xs:complexContent	定义 XML 元素或空元素

下列的程序段显示了只包含子元素的元素的定义，执行结果如图 5-6 所示。

```
⋮
<!--定义 studentlist 元素-->
<xs:element name="studentlist">
        <xs:complexType>
            <xs:sequence>
              <xs:group ref="studentType" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
</xs:element>
<!--  定义 studentType 组 -->
<xs:group name="studentType">
        <xs:sequence>
                <xs:element name="student" maxOccurs="1">
                 <xs:complexType>
                  <xs:sequence>
                    <xs:choice><!-- 属性 id 和 name 只能选择一个属性-->
                      <xs:element name="id" type="xs:integer"/>
                        <xs:element name="name" type="xs:string"/>
                    </xs:choice>
                    <xs:sequence>
                        <!--属性 birthday 和 gender 必须按顺序出现-->
                        <xs:element name="birthday" type="xs:date"/>
                        <xs:element name="gender" type="xs:string"/>
                    </xs:sequence>
                  </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
</xs:group>
⋮
```

```
<?xml version="1.0" encoding="UTF-8"?>
<studentlist xsi:noNamespaceSchemaLocation="groupchoice.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <student>
        <id>90343201</id>
        <birthday>1990-09-21</birthday>
        <gendar>男</gendar>
    </student>
    <student>
        <name>张珊</name>
        <birthday>1990-05-13</birthday>
        <gendar>女</gendar>
    </student>
</studentlist>
```

图 5-6　嵌套子元素的元素对应的 XML 片段

（2）包含子元素和文本的元素。如果要定义包含子元素有要定义元素本身内容的文本取值，就需要对 xs:complexType 的一个属性 mixed 进行说明。通常情况，xs:complexType 的 mixed 属性默认值为“false”，表示只定义 XML 元素。如果需要包含文本和子元素，则要将 mixed 属性设定为“true”。下列程序段显示了这一应用。

```
⋮
<xs:element name="student">
 <xs:complexType mixed="true">
    <xs:sequence>
    <xs:element name="name"/>
    <xs:element name="birthday" type="xs:date"/>
    <xs:element name="gendar" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
⋮
```

```
<student>
  90123
  <name>王一</name>
  <birthday>1990-12-10</birthday>
  <gendar>男</gendar>
</student>
```

图 5-7　包含子元素和文本的元素

(3) 空元素。空元素是不包含文本内容的元素。用 XML Schema 实现 XML 元素的定义是要利用 xs:complexContent 元素来实现的。xs:complexContent 元素是 xs:complexType 的子元素，通常是用于 xs:complexType 元素的扩展或限制。该元素常见的有两个属性。

① 属性 id：表示为元素指定唯一的编号；

② 属性 mixed：用于定义混合内容。默认值为 false，表示元素内容不允许出现在子元素中，如果设定 true，表示子元素可以出现元素内容。

下列程序段定义了一个带有属性的 XML 元素，执行结果如图 5-8 所示。

```
⋮
<xs:element name="student">
  <xs:complexType>
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="id" type="xs:string"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
</xs:element>
⋮
```

```
<student id="1234234"/>
```

图 5-8　XML 空元素片段

也可以通过下列更简单的形式实现空元素的定义。

```
⋮
<xs:element name="student">
    <xs:complexType>
      <xs:attribute name="id" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

(4) 包含属性的元素。通常 XML 元素带有一些属性，带属性的 XML 元素可以通过 XML Schema 的 xs:simpleContent 元素来实现。xs:simpleContent 元素作为 xs:complexType 的子元素，通常用于扩展或限制带有文本内容复杂类型元素和简单类型的元素。为此，它有两个子元素 xs:extension 和 xs:restriction，这两个子元素的作用如下。

① xs:extension 元素用于元素的扩展。

② xs:restriction 元素用于限制元素,设置元素的约束条件。

下列的程序清单 5-13 定义了带有属性的元素,执行结果如图 5-9 所示。

程序清单 5-13:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="studentlist">
        <xs:complexType mixed="true">
            <xs:sequence>
                <xs:element name="student">
                    <xs:complexType>
                        <xs:simpleContent>
                            <!-- 设置元素属性 -->
                                <xs:extension base="xs:string">
                              <!-- 引用组属性-->
                                <xs:attributeGroup ref="stuAttr"/>
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
<!-- 定义属性组 stuAttr-->
    <xs:attributeGroup name="stuAttr">
        <xs:attribute name="id" use="required"/>
        <xs:attribute name="status" default="registered"/>
    </xs:attributeGroup>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<studentlist xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:noNamespaceSchemaLocation="elementwithattribute1.xsd">
  记录1:
  <student id="902323" status="registered">张[illegible]</student>
</studentlist>
```

图 5-9 对应的 XML 片段

在上述代码中通过 XML Schema 的 xs:attributeGroup 元素对多个属性编组,形成组属性。组属性可以作为整体一起引用。

5.3 CSS 显示 XML

用 XML 可以定义文件内容,这些内容最终是显示给用户浏览查看。CSS 层叠样式表(见第 3 章)是可以显示 XML 文件的一种常见方式。通常,利用外部 CSS 文件显示 XML

内容，充分体现了内容和显示分离的要求。具体做法是将定义显示样式的内容定义到一个独立的 CSS 文件中，然后在 XML 文件中利用下列的语法格式指定 CSS 文件。

```
<?xml-stylesheet type="text/css" href="CSS 文件"?>
```

其中，type 属性指明了样式的种类为“text/css”；href 属性指明了外部的 CSS 文件。

为了说明利用外部 CSS 文件显示 XML 文件的效果，将程序清单 5-14 的 XML5-14.xml 用程序清单 5-15 的 CSS5-15.css 显示。浏览器中显示 XML5-14.xml 的效果如图 5-10。

程序清单 5-14：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="CSS5-15.css"?><!-- 导入 css 文件-->
<studentlist>
    <student>
        <id>90123</id>
        <name>王一</name>
        <birthday>1990-12-10</birthday>
        <gender>男</gender>
    </student>
    <student>
        <id>90124</id>
        <name>刘萨</name>
        <birthday>1992-02-19</birthday>
        <gender>男</gender>
    </student>
</studentlist>
```

90123
王一 1990-12-10 男
90124
刘萨 1992-02-19 男

图 5-10　外部 CSS 显示 XML 文件

程序清单 5-15：

```
studentlist{  border: inset;
              background-color: #FF9900;
              width: 30% ;
              }
student      { display: block;
              border: #FFFFCC groove;
              text-align: center;
              font-size: larger;
              }
id            {height:50px;
              }
name         {text-decoration: none;
              text-align: center;
              font-family: 隶书;
              }
gender       { font-weight: bolder;
              }
```

5.4　XSLT 转换 XML

XSLT 表示 XML Stylesheet Language Transformation，表示 XML 样式表语言转换。它是基于 XML，具有 XML 的基本语法要求，是 XSL 扩展样式语言的一个组成。1999 年，XSLT 成为 W3C 标准。当前的版本是 XSLT 2.0，是 1999 年推出的 XSLT 1.0 的修正版。与 CSS 层叠样式表一样，XSLT 可以实现按照特定格式显示 XML 文件的内容。但不同在于 CSS 仅仅设置 XML 文件的格式，XML 文件的内容不变。而 XSLT 是将一个 XML 文件转换成另外一个格式的 XML 文件，文件的内容发生变化。XSLT 就好比一个模板，XML 文件按照模板进行处理。

图 5-11 展示了 XSLT 的工作原理。首先，XSLT 样式是定义将 XML 树转换的规则。值得注意的是这种规则的建立不能脱离 XPath（见 5.4.1 小节），这是因为利用 XPath 表达式实现 XML 数据的选择，这是转换的一个前提。按照 XSLT 定义将选中的 XML 片段转换成其他形式的文档如 HTML、XHTML、PDF、VoiceXML 等。其次，通过 XSLT 处理器按照 XSLT 定义的规则进行转换成指定格式的内容。常见的 XSLT 处理器有 IE 浏览器内置的 MSXML、Saxon、Xalan 等。XSLT 充分体现了 XML 数据定义和 XML 显示的分离，并为相同 XML 数据的多种形式的重现成为现实。

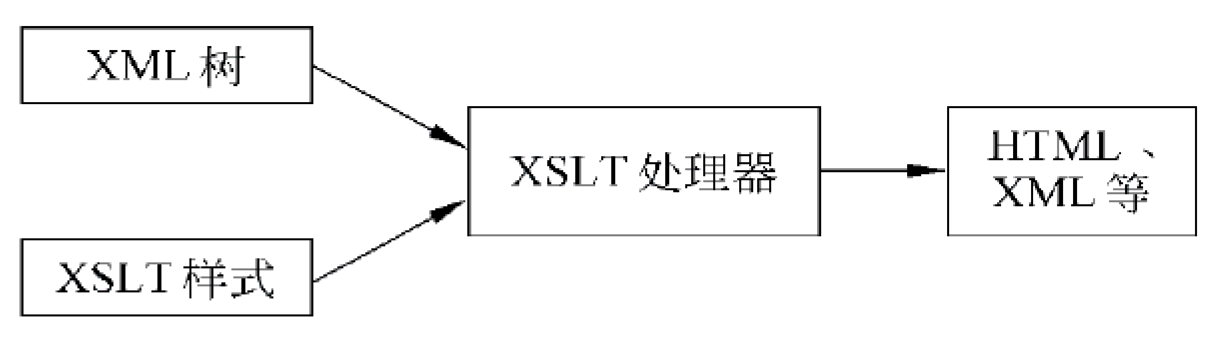


图 5-11　XSLT 的工作原理

5.4.1　XPath 基础

要了解 XSLT 转换 XML，必须从 XPath 开始。XPath 全称是 XML Path Language，表示 XML 路径语言。XPath 也是 XSL 的组成部分之一。XPath 的重要作用就是实现 XML 文件的导航。通过导航实现 XML 文件内部的定位，实现对 XML 文件的元素、属性、文本数据等内容的访问。但是作为 XSL 的组成，XPath 并不是基于 XML 的语法，而是采用类似目录结构的简洁语法内容。

在本节中，将从 XPath 的结点、XPath 位置路径、XPath 的轴以及 XPath 的运算符几个方面来了解 XPath 是如何实现对 XML 文件的导航。

1. XPath 的结点

XPath 将 XML 文档视之为结点树。结点树中由结点构成，各个结点可以表示 XML 文档的元素或属性或其他内容。为了定义 XML 的树状结构，XPath 定义了 7 种结点，具体内容见表 5-8。

注意：根结点并不对应于 XML 文件的根元素，而是根结点包含了根元素。

XPath 的结点之间并不是孤立的，而是存在特定的关系。XPath 将结点的关系定义为轴。了解如下几种关系，以程序清单 5-14 为例进行说明。

表 5-8　XPath 的结点

名　　称	说明(以程序清单 5-14 为例)
根(文档)结点	表示 XML 文件的层次结构的顶层,如 studentlist
元素结点	表示 XML 元素,如 student,id,name,birthday,gender
属性结点	表示 XML 元素的属性
文本结点	表示 XML 文件的文本数据内容
命名空间结点	表示 XML 命名空间前缀/URI 对
处理指令结点	表示 XML 文件的处理指令
注释结点	表示 XML 文件的注释

(1) 父(Parent)。每一个元素和属性都有一个父。例如:studentlist 是 student 的父,student 是元素 id、name、birthday、gender 和属性 status 的父。

(2) 子(Children)。元素结点的分支。例如:student 是 studentlist 的子,id、name、birthday 是 student 的子。

(3) 兄弟(Sibling)。具有相同父结点的结点。例如 id、name、birthday 和 gender 是兄弟关系。

(4) 祖先(Ancestor)。结点的前续结点。例如 id 的祖先有 student 和 studentlist。

(5) 后代(Descendant)。结点的后续结点。例如 studentlist 的后代有 student、id、name 等。

2. XPath 的位置路径(Location Path)

位置路径是 XPath 的重要语法内容。位置路径是 XPath 的表达式,可以描述其他结点相对于一个起点(当前结点)的位置。位置路径的作用是从当前结点开始,选择满足条件相对于起点位置的相关结点或结点集。XPath 有绝对定位和相对定位两种方式。绝对定位表示从根结点开始定位的方式,在设置时需要使用运算符“/”。而相对定位是当前结点开始的定位方式,利用运算符“//”实现。此外,XPath 定义了位置路径的运算符,见表 5-9。通过这些运算符实现选择满足特定条件的结点。

表 5-9　位置路径的运算符

运算符	说　　明
/	从根结点开始
//	从所在结点开始选择所有的后辈结点
.	表示当前结点
..	表示父结点
*	表示所有元素结点
@	表示元素的属性,@ * 表示元素的所有属性
node()	表示任何类型的结点
[]	选择运算,内含谓词,谓词表示 XPath 的导航特定条件
|	组合位置路径,实现多个路径

上述的运算符组合,可以形成 XPath 路径表达式。通过 XPath 的路径表达式来导航特定条件的结点或结点集。为了说明这些运算符的应用,结合图 5-12、程序清单 5-16 和表 5-10 来了解 XPath 表达式以及表达式的作用。

程序清单 5-16:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XML5-16.xml-->
<list>
  <student>
    <name>张山</name>
    <gender>男</gender>
    <school><name>×××市中心学校</name>
    </school>
  </student>
  <student>
    <name>李司</name>
    <gender>男</gender>
    <school><name>×××市西区学校</name></school>
  </student>
</list>
```

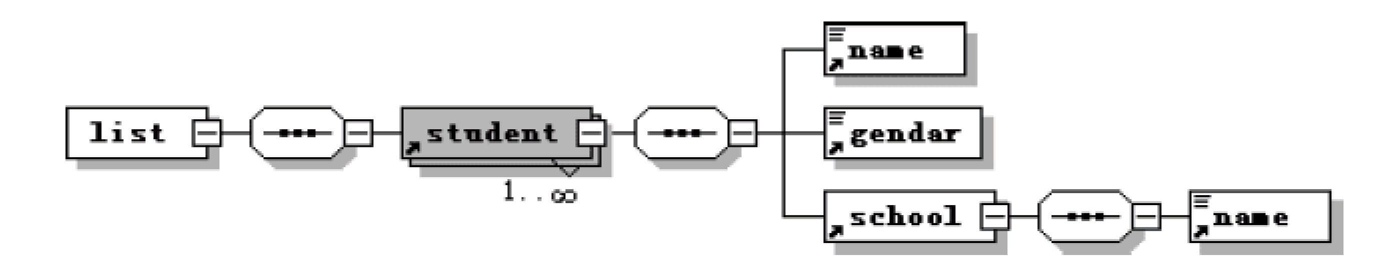


图 5-12 XML 文件的结构示意

表 5-10 XPath 的路径表达式举例

路径表达式	说　　明
/list/student	选择 list 下层 student 元素的所有子元素
//name	选择所有的 name 元素,包括 student 的子元素 name 和 school 的子元素 name
//student/name	选择父结点是 student 的 name 元素,即 student 元素的 name 子元素
//student/name|//school/name	选择所有 student 元素的 name 子元素和 school 元素的 name 子元素
/*/*/name	选择所有的有两个祖先的 name 元素,即 student 的子元素 name
/list/student[last()]	选择最后一个 student 元素

3. XPath 的轴(Axes)

XPath 查找 XML 元素时,从当前元素结点出发,可以向下步进到元素的子元素以及后代元素,也可以向上步进到元素的祖先,也可以侧向步进到元素的属性。如果当前结点是属性结点,也可以从属性结点步进到元素。这些步由 3 个部分构成:轴、谓词(见表 5-9)和结点测试,形式如下:

```
轴名称::结点测试[谓语]
```

(1) 轴：体现了其他结点与当前结点之间的关系。

(2) 结点测试：用于识别轴的内部结点。

(3) 谓词：使用任意表达式表示满足条件的结点集。

轴是其中最重要的部分，指明了结点步进的方向，是 XPath 导航的前提。XPath 定义了如表 5-11 所示的主要轴。

表 5-11　XPath 的轴

名　　称	说　　明	例　　子
self	当前结点	self::text()选取当前结点的所有文本
child	当前结点的子结点	child::node()选取当前结点的所有子结点
parent	当前结点的父结点	parent:: * 选取当前结点的所有父结点
descendant	当前结点的后代结点	descendant::name 选取当前结点的所有 name 的后代结点
descendant-or-self	当前结点后代结点及其当前结点本身	descendant-or-self:: * 选取当前结点及所有祖先结点
following	当前结点的后继结点	following:: * 选取当前结点的所有后代结点
following-sibling	当前结点兄弟结点的之后的所有结点	following-sibling:: * 选取当前结点的所有兄弟结点
preceding	当前结点之前的所有结点	preceding:: * 选取当前结点之前的所有结点
preceding-sibling	当前结点的兄弟结点的之前的所有结点	preceding-sibling:: * 选取当前结点之前的所有兄弟结点
attribute	当前结点的属性结点	attribute::status 选取当前结点的 status 属性结点
namespace	当前结点的命名空间结点	

为了能对 XML 文件进行准确导航，XPath 还提供了一些函数。在本节中提供了常见的核心函数，具体内容见表 5-12。

表 5-12　XPath 的常见核心函数

类别	函　　数	返回值	说　　明
结点集操作	last()	数字	返回最后一个结点的位置
	position()	数字	返回表达式指定当前结点的位置
	count(结点集)	数字	计算结点集中的结点数
	id(对象)	结点集	用对象的代码号选取元素
	name(结点集)	字符串	返回结点集中第一结点名称
	local-name(结点集)	字符串	返回结点集中第一结点名称(无命名空间前缀)
	namespace-uri(结点集)	字符串	返回结点集中第一结点的命名空间 URI

类别	函　　数	返回值	说　　明
字符串操作	string(对象)	字符串	将参数中的对象转换成字符串
	concat(字符串,字符串,字符串)	字符串	连接字符串
	starts-with(字符串 1,字符串 2)	布尔值	判断字符串 1 开始字符串是否是字符串 2
	contains(字符串 1,字符串 2)	布尔值	判断字符串 1 是否包含字符串 2
	substring-before(字符串 1,字符串 2)	字符串	取字符串 1 中字符串 2 之前的子串
	substring-after(字符串 1,字符串 2)	字符串	取字符串 1 中字符串 2 之后的子串
	substring(字符串,数字 1,数字 2)	字符串	取参数中的字符串从数字 1 到数字 2 之间的子串
逻辑操作	boolean(对象)	布尔值	将参数转换成布尔值
	not(布尔值)	布尔值	非运算
	true()	布尔值	返回假值
	false()	布尔值	返回真值
	lang(字符串)	布尔值	
数字操作	number(对象)	数字	将参数转换成数字
	sum(结点集)	数字	返回结点集的各结点的文本结点值总数
	floor(数字)	数字	返回不大于参数的最大整数
	ceiling(数字)	数字	返回不小于参数的最小整数
	round(数字)	数字	四舍五入为整数

现在以程序清单 5-16 的 XML 文件为例,如果要求选取以“s”开头的所有结点,则用 XPath 可以表示如下形式:

```
//* [start-with(name(),'s')]
```

将 XPath 表示的树图可以用图 5-13 表示,其中灰色背景显示的结点是满足上述的选取条件。

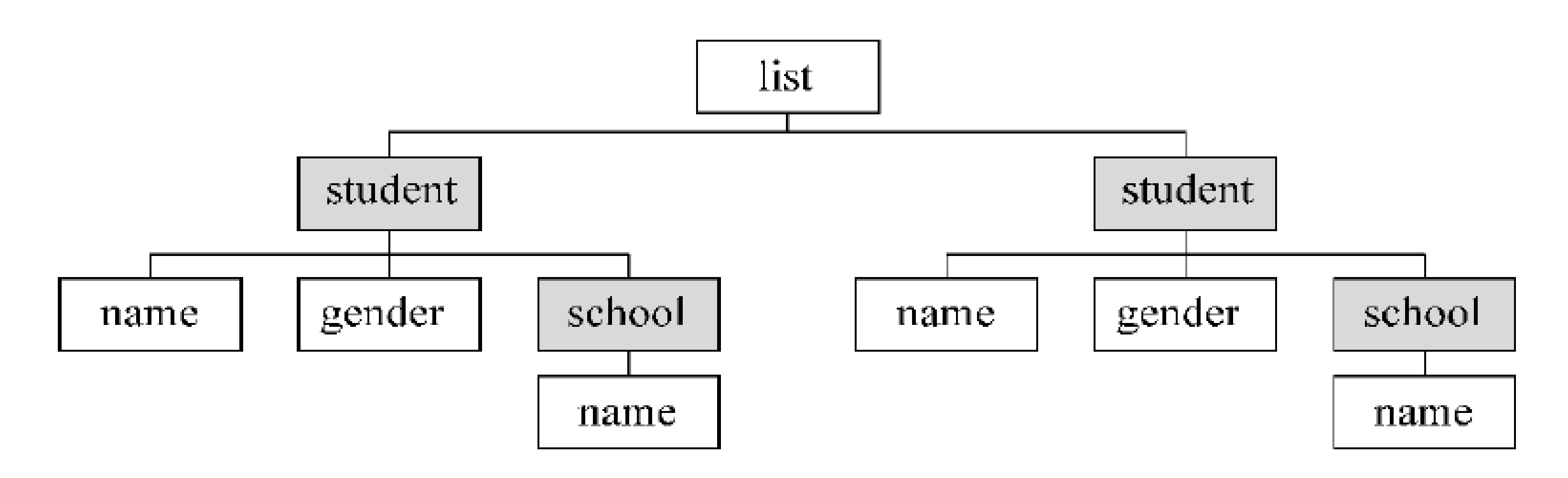


图 5-13　对应 XPath 表示形式的树图

5.4.2　XSLT 的基本结构

XSLT 可转换 XML 数据,将 XML 数据转换成其他形式的文档。与 XPath 不同在于,

XSLT 是基于 XML 的，它具有 XML 的语法要求。作为转换 XML 的一种样式表定义形式，XSLT 也具有其自身的特点。在本节中，将探讨 XSLT 的基本结构，及组成 XSLT 的主要元素。

1. XSLT 基本结构

XSLT 的基本结构如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version=版本号 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    ...<!-- 加入 XSLT 元素 -->
</xsl:stylesheet>
```

首先，XSLT 是基于 XML。所以，在 XSLT 文件中，必须用“<? xml version="1.0">”先说明为 XML 文件。然后，为了实现创建一个 XSLT 样式表，可以用“<xsl:stylesheet>”元素或“<xsl:transform>元素”来说明。无论是那一种元素，它们的“version”和“xmlns:xsl”属性是定义 XSLT 必不可少的。“version”属性指明转换样式表的版本为“XSLT 1.0”，而“xmlns:xsl”属性指定 XSLT 的命名空间。在“xsl:stylesheet”或“xsl:transform”元素之间，是一些实现具体转换功能的元素。

2. XSLT 元素

除了用于创建 XSLT 之外样式表元素“<xsl:stylesheet>”，XSLT 还具有预先定义了具有特定功能的其他元素。这些元素见表 5-13。

表 5-13　XSLT 常见元素

XSLT 元素	说　　明
<xsl:stylesheet>	样式表定义元素
<xsl:transform>	样式表定义元素，作用同<xsl:stylesheet>
<xsl:template>	定义模板，制定选取结点的转换规则
<xsl:apply-templates>	调用模板使之适用于当前结点或子结点
<xsl:call-template>	调用一个已命名的模板
<xsl:value-of>	抽取选取结点的值
<xsl:element>	创建一个元素
<xsl:attribute>	增加属性
<xsl:attribute-set>	定义已命名的属性集
<xsl:comment>	创建注释结点
<xsl:text>	输出文本
<xsl:for-each>	循环，在特定结点集中依次遍历每一个结点
<xsl:choose> <xsl:when> <xsl:otherwise>	这 3 个元素组合，实现多条件判断
<xsl:if>	条件判断，条件满足才执行已包含的规则

XSLT 元素	说　明
<xsl:copy>	创建当前结点的复本(不包括子结点和属性)
<xsl:copy-of>	创建当前结点的复本(包括子结点和属性)
<xsl:import>	将一个优先级低的样式表的内容导入到另外一个样式表
<xsl:include>	将一个样式表的内容包含到另外一个样式表(优先级相同)
<xsl:output>	定义输出格式
<xsl:sort>	排序输出
<xsl:param>	定义变量
<xsl:with-param>	指定参数的值

(1) <xsl:template>元素。“xsl:template”元素是模板元素,用于定义 XML 元素的转换规则。“xsl:template”元素有一个重要属性“match”。这个属性的作用是利用 XPath 表达式指出需要套用模板的 XML 元素的位置路径。对于符合 XPath 导航条件的位置路径,则需要按模板的定义进行转换。

(2) <xsl:value-of>元素。“xsl:value-of”元素是内容元素。该元素的最主要作用是获取 XML 数据(包括 XML 元素和属性)的具体内容。与“xsl:template”一样,“xsl:value-of”元素需要 XPath 来定位 XML 数据。具体实现是先用 XPath 定位 XML 数据的位置路径,“xsl:value-of”元素通过属性有“select”选择位置路径指定的相关数据。

【例 5-1】 要求编写一个 XSLT 文件,能将程序清单 5-9 中的第二个 email 相关数据按照 XHTML 的表格形式显示,执行结果如图 5-14 所示。

程序清单 5-17:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XSLT5-17.xslt-->
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">          <!--对根结点下的元素匹配-->
   <html>
      <head>
         <title>内容显示</title>
      </head>
      <body>
      <table border="1">
        <tr>
          <th>编号</th>
          <th>from</th>
          <th>to</th>
          <th>主题</th>
          <th>信息内容</th>
        </tr>
        <tr>                    <!--显示第 2 个 email 位置路径下的相关信息-->
```

```
            <td><xsl:value-of select="mailbox/email[position()=2]/@id"/></td>
            <td><xsl:value-of select="mailbox/email[position()=2]/from"/></td>
            <td><xsl:value-of select="mailbox/email[position()=2]/to"/></td>
            <td><xsl:value-of select="mailbox/email[position()=2]/subject"/></td>
            <td><xsl:value-of select="mailbox/email[position()=2]/message"/></td>
          </tr>
        </table>
    </body>
    </html>
    </xsl:template>
</xsl:stylesheet>
```

编号	from	to	主题	信息内容
liu2	wang@yahoo.com.cn	wang_he@yahoo.com.cn	问好	假期回家

图 5-14　指定元素的转换实例

（3）控制元素。XSLT 与其他语言一样也提供了控制元素。这些控制元素有循环元素“xsl:for-each”、单条件元素“xsl:if”和多条件元素“xsl:choose”。

循环元素“xsl:for-each”可以将拥有相同条件的 XML 数据进行循环转换。“xsl:for-each”元素也有“select”属性，该属性“select”必须和 XPath 结合确定需要转换 XML 数据的范围。

单条件元素“xsl:if”可以测试 XML 数据是否满足转换的条件。其中，具体条件是通过该元素的 test 属性进行设置。如果 test 属性中规定的条件满足，则按照定义的规则进行转换，否则不实现转换处理。

当然，单条件元素对于一些多条件处理情况存在局限，XSLT 推出的“xsl:choose”多条件元素可以解决这些不足。“xsl:choose”元素常常与“xsl:when”和“xsl:otherwise”结合使用。具体应用的语法形式如下：

```
<xsl:choose>
  <xsl:when test="条件 1">
     ⋮
  </xsl:when>
  <xsl:when test="条件 2">
     ⋮
  </xsl:when>
     ⋮
  <xsl:when test="条件 n">
     ⋮
  </xsl:when>
  <xsl:otherwise>
     ⋮
  </xsl:otherwise>
</xsl:choose>
```

【例 5-2】　要求编写一个 XSLT 文件，能将程序清单 5-9 中所有 wang@yahoo.com.cn

发送的 email 的相关数据按照 XHTML 的表格形式显示，执行结果如图 5-15 所示。

程序清单 5-18：

```
<?xml version="1.0" encoding="UTF-8"?>
<!--XSLT5-18.xslt-->
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">                        <!--对根结点下的元素匹配-->
    <html>
        <head>
            <title>内容显示</title>
        </head>
        <body>
        <table border="1">
          <tr>
            <th>编号</th>
            <th>from</th>
            <th>to</th>
            <th>类别</th>
            <th>主题</th>
            <th>信息内容</th>
          </tr>
          <xsl:for-each select="mailbox/email">          <!--循环遍历 email 元素-->
          <xsl:if test="from='wang@yahoo.com.cn'">    <!--判断 from的值-->
          <tr>
            <td><xsl:value-of select="@id"/></td>
            <td><xsl:value-of select="from"/></td>
            <td><xsl:value-of select="to"/></td>
            <td><xsl:value-of select="to/@catalog"/></td>
            <td><xsl:value-of select="subject"/></td>
            <td><xsl:value-of select="message"/></td>
          </tr>
          </xsl:if>
          </xsl:for-each>
        </table>
    </body>
    </html>
    </xsl:template>
</xsl:stylesheet>
```

编号	from	to	类别	主题	信息内容
liu2	wang@yahoo.com.cn	wang_he@yahoo.com.cn	family	问好	假期回家
liu3	wang@yahoo.com.cn	he_jiang@yahoo.com.cn	company	请假批准	春假批准，时间是2月1日至2月20日。

图 5-15　有循环和条件判断的转换实例

（4）＜xsl:apply-templates＞元素。“xsl:apply-templates”元素是递归调用模板元素。

该元素的作用是将定义的模板递归调用到当前结点以及子结点。该元素有一个可选属性 select，该属性设置指定结点的转换规则。通过 select 属性也可以规定模板转换的顺序。

【例 5-3】 要求编写一个 XSLT 文件，能将程序清单 5-9 中所有 email 元素的 from 与 to 元素按照列表形式输出显示，执行结果如图 5-16 所示。

程序清单 5-19：

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!--XSLT5-19.xslt-->
    <xsl:template match="/"><!--对根结点下的元素匹配-->
    <html>
        <head>
            <title>内容显示</title>
        </head>
        <body>
            <xsl:apply-templates/><!--应用模板-->
        </body>
    </html>
    </xsl:template>
    <xsl:template match="email"><!--定义 email 元素模板-->
        <ul>
            <li><xsl:apply-templates select="from"/></li><!--应用 from 元素模板-->
            <li><xsl:apply-templates select="to"/></li><!--应用 to 元素模板-->
        </ul>
    </xsl:template>
    <xsl:template match="from"><!--定义 from 元素模板-->
      <span style="color:#00ff00"><xsl:value-of select="."/></span>
    </xsl:template>
    <xsl:template match="to"><!--定义 to 元素模板-->
      <b><xsl:value-of select="."/></b>
    </xsl:template>
</xsl:stylesheet>
```

- liu@yahoo.com.cn
- **wang@yahoo.com.cn**

- wang@yahoo.com.cn
- **wang_he@yahoo.com.cn**

- wang@yahoo.com.cn
- **he_jiang@yahoo.com.cn**

图 5-16 递归调用模板的转换实例

5.4.3 用 XSLT 显示 XML

实际上 XSLT 之所以可以显示 XML 文件，这是因为 XSLT 可以将 XML 文件转换成特定格式的文档，然后以这些文档形式进行显示。并不同于 CSS 的为 XML 文件设置特定的显示格式。为了实现 XSLT 转换 XML 文件，需要在 XML 文件中声明使用 XSLT 定义的样式表，具体定义形式如下：

```
<?xml-stylesheet type="text/xsl" href="XSLT样式表文件"?>
```

其中：

（1）type 属性指定了样式表的格式为“text/xsl”，表示使用 XSLT 的样式表；

(2) href 属性明确了样式表文件的 URL，按照指定的位置使用 XSLT 样式表。

将下列程序清单 5-20 定义的 XML 文件，用程序清单 5-21 定义的 XSLT 样式表来进行显示，执行结果如图 5-17 所示。

程序清单 5-20：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xslt5-21.xslt "?>
<!-- xml5-20.xml -->
<studentlist>
  <student>
        <id>90123</id>
        <name>王一</name>
        <birthday>1990-12-10</birthday>
        <gender>男</gender>
  </student>
  <student>
        <id>90124</id>
        <name>刘萨</name>
        <birthday>1992-02-19</birthday>
        <gender>男</gender>
  </student>
</studentlist>
```

程序清单 5-21：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- xslt5-21.xslt -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
        <div>
            <xsl:for-each select="studentlist/student"><!--选择 student 子元素-->
            <b><xsl:value-of select="name"/></b>
            <br/>
            <xsl:value-of select="id"/>
            <xsl:text>   </xsl:text><!--空格-->
            <xsl:value-of select="birthday"/>
            <xsl:text>   </xsl:text><!--空格-->
            <xsl:value-of select="gender"/>
            <br/>
            </xsl:for-each>
        </div>
  </xsl:template>
</xsl:stylesheet>
```

王一
90123 1990-12-10 男
刘萨
90124 1992-02-19 男

图 5-17　XML 文件运行结果

小　　结

XML 是可扩展标记语言，常用于数据的表达。它具有良构性、可扩展性、灵活性和简单性。符合当前的数据表达与表现分离的基本原则。本章从 XML 客户端应用出发，详细介绍了 XML 的基础知识。特别对于 XML 的语法要求，以及建立一个良构的 XML 文件必须遵循的原则进行深入浅出的说明。为了更加符合 XML 应用的要求，本章还介绍了 XML 验证两种机制：文档类型定义和 XML 模式定义语言，以及 XML 的两种显示方式：CSS 显示 XML 与 XSLT 转换 XML。XML 实质是一个服务器端的技术，更深入的介绍见第 14 章。

练　习　5

1. **填空**

(1) XML 是________语言，它具有________、________、________和________特点。

(2) ________和________用于 XML 文件的验证。

(3) XPath 表示________。

(4) ________和________可以以 HTML 网页形式显示 XML 文件。

(5) 在 HTML 文件中声明 XML 数据岛的语法形式是________。

2. 选择题

(1) DTD 表示________

A. 文件定义类型　　B. 文档类型定义　　C. 定义类型文件

(2) CSS 表示________

A. 层叠样式表　　B. HTML 样式表

C. XML 显示样式表　　D. XML 转换样式表

(3) 下列 XSLT 片段可以实现选取________数据。

```
⋮
<xsl:value-of select="/mailbox/email[position()=2]/@id"/>
⋮
```

A. 提取 mailbox 结点的孩子 email 结点的第 2 个孩子结点的 id 属性的内容；

B. 提取 mailbox 结点的孩子 email 结点的第 2 个孩子结点的 id 元素的内容；

C. 提取 mailbox 结点的第 2 个 email 孩子结点 id 属性的内容；

D. 提取 mailbox 结点的第 2 个 email 孩子结点 id 元素的内容；

3. 实验题

设计一个个人简历 XML 文件，用 XML Schema 机制进行验证。最后用 XSLT 以良好的表格形式进行显示。

第 6 章　WAP 2.0 编程

6.1　WAP 2.0 简介

随着无线移动手持设备(如手机、个人数字助理 PDA 等)的广泛应用和普及，一些用户对这些设备提出新的应用要求也应运而生。实现这些新应用的后面是无线移动应用技术的不断更新和发展。当前主流的移动技术有 WAP、J2ME 和 Brew 等。WAP 是其中的佼佼者，得到了当前大部分移动受限手持设备的支持。

WAP 是 Wireless Application Protocol 的简称，表示无线应用协议。WAP 是沟通了无线世界和互联网的一种通信协议和应用环境的技术标准。WAP 为无线移动设备提供了互联网中内容和高级的数据服务。WAP 是适应于无线移动设备访问 Internet 而产生的。

众所周知，无线移动设备是一种典型的受限设备。例如手机，手机为了携带方便，尺寸小，手机的显示屏有限、显示分辨率小，手机存储能力弱，手机主要靠电池运行，手机处理器的计算能力也非常有限，手机的键盘的输入能力无法与台式机的键盘相比。这些限制条件的存在，在很长时间内制约了无线移动设备的网络访问功能，决定了无线移动设备无法(如台式计算机)访问互联网。但是伴随着广大的无线应用要求，急切需要解决受限移动设备无法访问互联网问题。在 1997 年 6 月，由"Ericsson"、"Nokia"、"Motorola"和"Phone. com"几家公司出面成立了一个 WAP 论坛(WAP Forum)，由 WAP 论坛来制定和改进 WAP 技术标准。WAP 问世的最主要的目的是让无线移动设备不受生产厂家、网络基础设施等条件的限制而实现无线互联网的访问。

1998 年 4 月，WAP 论坛推出了 WAP 1.0 版本。在 WAP 1.0 版本中充分体现了多层次协议标准的特点。WAP 1.0 中吸收了"Nokia"的"窄带套接字"(NBS)和"标签文本标记语言"(LTTL)、"Ericsson"的"智能终端传输协议"(ITTP)和"Unwired Planet"的"手持设备标记语言"(HDML)的技术特点。在随后的几年里陆续推出了 WAP 1.1、WAP 1.2。

但是，WAP 1. x 的问世并没有带来市场的广泛应用。在此前提下，WAP 论坛对 WAP 标准进行改进。2002 年，WAP 论坛与 Open Mobile Architecture Initiative 两个组织合并为 Open Mobile Alliance(OMA)，OMA 推出的 WAP 2.0 版本。随着技术的发展，WAP 2.0 也不断修正更新。WAP 2.0 的主要特点如下。

(1) 支持通信协议广泛。WAP 2.0 采用双协议架构。它不但支持 WAP 1.2 所支持的 WSP/WTP/WDP 协议，而且当前 WAP 2.0 支持 HTTP、TCP/IP 和 TLS 协议。一方面，WAP 2.0 保证继续兼容 WAP 1. x；另一方面，WAP 2.0 将互联网协议引入 WAP 应用，充分利用现有的 Internet 技术。同时，由于 WAP 2.0 网关不需要对 WSP/WTP/WDP 转换成 HTTP/TCP 协议，极大地提高了运行速度和执行性能。

(2) WAP 2.0 的存在，使得无线移动设备突破不同网络环境制约。无论采用 2.5G 还是 3G 通信网络。无线移动设备都可以通过 WAP 2.0 来获取互联网的服务。

(3) WAP 2.0 提供了丰富的应用环境。WAP 2.0 的应用环境与互联网的标准日趋一致。由于 WAP 2.0 采用了与互联网兼容的技术,应用领域进一步扩大。更重要的是许多无线移动设备的厂商支持 WAP 2.0 浏览器。这就使得 WAP 2.0 应用得到广泛实现。

(4) 应用 WAP 2.0 技术能使无线移动设备消耗较低的电力。这更加便利各种无线移动设备之间通过互联网进行信息联系和交互。

6.2 WAP 2.0 的标记语言

WML 作为一个标记语言脱胎于"Unwired Planet"的 HDML,WAP 论坛在 1998 年将它发展成为 WML 1.0。WAP 论坛不断继续发展 WML 技术,随之而来依次推出 WML 1.1 、WML 1.2 和 WML 1.3。但是由于无线网络的带宽的限制,访问 WAP 网页会有较长的时间延迟,并没有为无线移动设备带来更为广泛的市场应用。

2001 年 7 月 WAP 论坛制定 WAP 2.0 协议,对 2000 年 WML 1.3 改进和完善,使 WML 1.3 的性能进一步提高,为向后兼容提供了很好的支持。但是,WML 1.x 中存在与 HTML 类似的问题,即:WML 1.x 既定义格式又定义内容,使得 WAP 网页难以理解,进而导致 WAP 网页的适用范围受到限制。

为了适应互联网发展要求,WAP 论坛开始与 W3C 联盟合作研究移动因特网技术标准。WAP 论坛根据 W3C 推荐 XHTML Basic 标准,于 2001 先后推出 WML 2.0 和 XHTML Mobile Profile(简称 XHTML MP)。XHTML MP 是 W3C 的 XHTML Basic 标准的严格子集,它在 XHTML Basic 基础上增加了部分额外的 XHTML 模块,主要提供对 CSS 和内容显示处理的支持。XHTML MP 的出现使得无论是 PC 还是受限移动设备都可以浏览 Internet 信息,使得无线和有线的标记语言开始走向融合。WML 2.0 是 XHTML MP 的扩展,吸收了当前互联网的主要技术标准。WML 2.0 的主要特性在于一方面向后与 WML 1.x 兼容,保留了 WML 1.x 同样的功能,另一方面与 XHTML 技术很好的结合,具有 XHTML 语言的特点。

开发用上述标记语言编写的 WAP 网页非常简单,可以使用任何一种文本编辑器即可。常见的开发工具有 Nokia Mobile Internet Toolkit 4.1。为了测试 WAP 2.0 网页,可以使用 WinWAP 4.0 for RC1、OpenWave SDK 对网页进行测试运行。在本节将对 WML 1.3、WML 2.0 以及 XHTML MP 做一个介绍。

6.2.1 无线标记语言 WML

WML 的全称是 Wireless Markup Language,表示无线标记语言,是 XML 1.0 的子集,具有 XML 的语法特点。WML 是开发 WAP 应用的一项重要技术,它定义了服务无线移动设备的一套无线标记库,通过标记库实现包括文本、图像、音频等内容的描述。WML 的作用与 HTML 类似,都是可以实现网页的设计与显示。不同在于 WML 是服务于无线移动设备,是在 WAP 浏览器中展示 WML 制作的 WAP 网页。本节中将采用 WML 1.3 展开介绍。

1. WML 的文件结构

WML 文件实际上是一个 XML 文件,只是文件名的后缀为".wml"。一个 WML 文件

的基本结构如下：

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
"http://www.wapforum.org/DTD/wml13.dtd">
<wml>
  <card id="卡片名 1" title="标题 1">
    <!--添加卡片 1内容-->
  </card>
  ⋮
  <card id="cardn" title="标题 2">
    <!--添加卡片 n内容-->
</card>
</wml>
```

WML 文件必须指定“<?xml?>”和“<! DOCTYPE>”。其中“<?xml?>”规定 XML 的版本以及使用的编码等相关内容。“<! DOCTYPE>”定义文档类型为 WML，指定 DTD 的类型。

WML 文件主体是由 wml 元素定义的卡片组。在卡片组内用 card 元素定义相互关联的不同的卡片。其中，id 属性表示不同的卡片，title 属性说明卡片的标题。在卡片组中必须包含至少一张卡片否则没有意义。卡片内部可以定义处理文本格式、布局、导航、链接、图像、任务和文本输入的相关标记，如表 6-1。通过这些 WML 1.3 的标记为卡片定义具体功能。

表 6-1　WML 1.3 的常见标记

类别	标　记	说　　明	类别	标　记	说　　明
文件结构	wml	定义卡片组	输入控制	select	定义列表
	card	定义卡片		optgroup	定义选项组
	head	定义 wml 文件头信息		option	定义列表项
	meta	描述文件元信息		input	定义文本输入框
	access	定义卡片组访问控件信息		fieldset	文本域，将嵌套的文本和输入框作为整体
	template	定义所有卡片的模板			
文本格式化	b	字体粗体	任务	go	导航到特定 URL 任务
	big	大号字体		prev	导航到历史记录的前一个 URL 任务
	em	着重字体		refresh	更新当前显示任务
	i	斜体		noop	不执行任务
	strong	强调字体	事件	do	激活任务
	small	小号字体		onevent	定义事件处理器
	u	下划线		postfield	向服务器发送信息

类别	标　记	说　明	类别	标　记	说　明
文本布局	p	定义段落	锚点	anchor	定义锚点，必须与 go 元素结合
	pre	预定义文本		a	定义锚点，用 href 属性定义一个超链接
	br	换行	图像	img	指定图像，wml 支持的图片格式有 wbmp、jpg、gif、bmp、wpng 和 png 等
	table	定义表格			
	tr	定义表格行	变量	setvar	设置变量
	td	定义表格的单元格		timer	定义定时器，每张卡片只有一个计时器

2. WML 文本处理

WML 文件中可以对文本信息，进行相应的处理。可以设置文本的字体格式以及将一段文本信息按照定义形式布局。这些内容的处理与 HTML 中同名的标记功能类似。

【例 6-1】 定义一个 WAP 网页，可以显示所有 wml 字体格式的表格，如图 6-1 所示。



图 6-1　字体表格运行结果

程序清单 6-1：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
     "http://www.wapforum.org/DTD/wml13.dtd">
<!--wml6-1.wml-->
<wml>
  <head>
    <meta name="author" content="Chen Yi">
    <meta name="keyword" content="wml tutorial">
  </head>
  <card id="card1" title="First Card">
    <table><!--定义表格-->
      <tr>
        <td><big>大号字</big></td>
        <td><small>小号字</small></td>
      </tr>
      <tr>
        <td><strong>强调字</strong></td>
        <td><em>着重字</em></td>
      </tr>
      <tr>
        <td><u>下划线</u></td>
        <td><b>粗体字</b></td>
      </tr>
```

```
            <tr>
              <td><i>斜体字</i></td>
            </tr>
          </table>
        </card>
      </wml>
```

3. WML 的输入

WML 1.3 提供了各种形式的输入的支持。实现客户对信息输入的标记有 input、fieldset、select、optgroup、option 这 5 个元素。input 定义文本框、select 与 optgroup、option 结合定义选择列表。至于 fieldset 是定义文本域，对卡片组中的文本框与选择列表等相关元素进行分组。

(1) input 元素。WAP 页面通过 input 元素接收客户的信息。input 元素的常见属性如下。

① name：是文本框名，用来区别不同的文本输入框，是必须定义的属性。

② type：定义文本框的类别。主要有两种类别，如果 type 属性默认或 type 的值为 text 则定义一般文本输入框；如果 type 值为 password，则为密码框，接收客户输入文本信息将以密文形式显示。

③ format：确定输入文本的格式。这些格式通过特定的符号来表示，具体内容见表 6-2。

表 6-2　应用于 format 的格式符

格式符	含　　义
A，a	表示字母、标识符、标点符号；字母 A 表示大写，a 表示小写
N	表示数字
n	表示数字、标识符、标点符号
X，x	表示字母、标识符、标点符号或数字；其中，字母 X 表示大写，x 表示小写
M，m	当前语言下的合法字符。如果语言支持区分大小写，M 则取默认字母为大写，m 则取默认字母为小写
* f ，nf	取 * 任何长度的字符，如果取 n(数字)表示取指定个数的字符。其中 f 是上述格式符的一种
\c	将斜杠后的字母原样输出，例：NNN\-3N 表示字符串形如"234-454"

④ size：指定文本框的字段的宽度。

⑤ emptyok：表示文本框是否是必须输入。如果 emptyok 的属性为 false，表示文本框不能为空；为 true，文本框可以忽略。默认取值为 false。

⑥ value：指定文本框的默认值。

⑦ maxlength：指定允许客户输入的最大字符数。

【例 6-2】　文本输入框的简单实例，实现用户名与密码的输入界面。

程序清单 6-2：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
      "http://www.wapforum.org/DTD/wml13.dtd">
<!--WML6-2.wml-->
<wml>
   <card id="card1" title="Card # 1">
      <p>
      用户名<br/>
      <input name="uname" emptyok="true" format="M* m">
      密码<br/>
      <input name="pword" type="password" emptyok="true">
      </p>
   </card>
</wml>
```



图 6-2　文本输入框的运行结果

（2）select、option 和 optgroup。这 select、option、和 optgroup 元素用来定义各种形式列表，通过客户选择列表中选项，来实现数据的输入。select 表示声明列表。select 有一个重要属性 multiple，通过它设置列表类型。当 multiple 为默认值或取值为 false 表示为单选列表；如果 multiple 取值为 true 定义一个复选列表。option 定义列表的选择单项。option 用属性 value 定义选项的值。用 onpick 属性定义一个选择事件，表示选择该项将转向的 URL。optgroup 元素可以设置多个选项成组，将组内的选项作为整体进行处理。

【例 6-3】 用选择列表实现题型选择的简单实例，如图 6-3～图 6-5 所示。



图 6-3　具有项目组选择列表运行结果



图 6-4　单选列表运行结果



图 6-5　复选列表运行结果

程序清单 6-3：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
      "http://www.wapforum.org/DTD/wml13.dtd">
```

```
<!--WML6-3.wml-->
<wml>
   <card id="card0" title="菜单">
     <p>
      <select name="type">
       <optgroup title="题型选择">
         <option value="1" onpick="#card1">
          单选题
        </option>
         <option value="2" onpick="#card2">
          多选题
        </option>
       </optgroup>
      </select>
     </p>
   </card>
   <card id="card1" title="单选题">
     <p>单选题:<br/>世界上最高的山峰<br/>
      <select name="answer1">
        <option value="A">珠穆朗玛峰</option>
        <option value="B">马卡鲁峰</option>
      </select>
     </p>
   </card>
   <card id="card2" title="多选题">
     <p>多选题<br/>多瑙河流经的国家?<br/>
      <select multiple="true">
        <option value="A">A.德国</option>
        <option value="B">B.奥地利</option>
        <option value="C">C.斯洛伐克</option>
        <option value="D">D.英国  </option>
      </select>
     </p>
   </card>
</wml>
```

4. WML 任务和事件

WML 的任务与 WML 事件密切相关。WML 的事件包含了两层意思就是检测动作发生和具体要执行动作。WML 支持四种类型的事件。

(1) ontimer 事件,定义计时器处理事件。

(2) onenterforward 事件,定义向前处理事件。

(3) onenterbackward 事件,定义向后处理事件。

(4) onpick 事件,定义选择事件处理。通常,WML 用 onevent 元素中声明并绑定具体的事件类型。用 do 元素激活当前用户执行具体任务。

WML 的任务是指示浏览器执行的事件发生的动作。WML 的任务按照作用域有两种情况:一种是与针对卡片组中所有卡片的任务,通常定义在 template 元素内;另一种是针对特定卡片的任务,在 card 元素内定义。

但从具体执行的动作而言,WML 中定义了 4 种任务 go、prev、refresh 和 noop。go 任

务是转换指定位置卡片，是用 go 标记来定义的。go 标记用 href 属性指定要转向卡片的 URL。形式如下："<go href="URL"/>"。prev 任务是定义转向历史堆栈的前一次访问的卡片。refresh 任务就是用 refresh 元素定义刷新当前卡片，修改卡片的变量。noop 任务表示一种特殊的任务就是什么也不执行。

【例 6-4】 WML 任务与 WML 事件的简单实例，设计一个 WAP 网页，先显示欢迎信息，5 秒后进入下一页面，执行结果如图 6-6 和图 6-7 所示。



图 6-6　第一张卡片



图 6-7　5 秒后的第二张卡片

程序清单 6-4：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
     "http://www.wapforum.org/DTD/wml13.dtd">
<!--WML6-4.wml-->
<wml>
  <template>
    <do type="accept" label="返回"><!--卡片组 prev 任务-->
       <prev/>
    </do>
  </template>
  <card id="card1" title="第一张卡片">
     <onevent type="ontimer"><!--定义计时器事件-->
      <go href="#card2"/>
     </onevent>
     <timer value="50"/><!--计时 5秒-->
     <p align="center">
      <img src="welcome.wbmp"/><br/>
        欢迎学习 WML!
     </p>
  </card>
  <card id="card2" title="第二张卡片">
    <p align="center">
      <big><b>WML学习内容</b></big>
```

```
    </p>
  </card>
</wml>
```

5. WML 变量

与 XHTML 网页一样，WAP 网页也需要在卡片组中的不同卡片中传递数据。WML 定义变量来实现这样的数据传输。WML 变量实际上就是代表一定值的字符串。字符串表示变量名，与其他语言一样，WML 变量具有一定的命名规则。

（1）WML 变量是由 0～9 的数字、A～Z 和 a～z 大小写字母与下划线字符组成。其他字符是非法命名字符，不能作为变量名。

（2）WML 的变量名区分大小写。

WML 变量可以通过 3 种形式获得：

（1）WML 变量可以是用 setvar 元素指定的变量，形式如下："<setvar name="变量名" value="值"/>"，name 属性指定变量名，value 属性指定变量具有的值。setvar 元素必须与 go、prev 和 refresh 任务结合使用。

（2）input 输入框中获取变量。

（3）从 select 定义的选择列表中获取变量。设置变量的目的是为了使用变量。在 WML 中用形如"$(变量名)"来获取变量。

注意：由于"$"作为取变量运算，所以在页面中输出"$"符，请用"$ $"来实现。

【例 6-5】 用 WML 变量显示用户输入的信息，运行结果如图 6-8 和图 6-9 所示。

程序清单 6-5：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
     "http://www.wapforum.org/DTD/wml13.dtd">
<!--WML6-5.wml-->
<wml>
  <card id="card1" title="Card #1">
    <do type="accept" label="下一张">
          <go href="#card2"><setvar name="varid"  value="编号:43" />
          </go>
    </do>
    <p align="center">
      <big><b>用户信息</b></big><br/>
      用户名:<br/>
      <input name="username"/>
      密码:<br/>
      <input name="password" type="password" />
    </p>
  </card>
  <card id="card2" title="Card #2">
    <p align="center">
      <big><b>显示信息</b></big><br/>
      用户名=$(username)<br/>
      密码=$(password)<br/>
       code=$(varid)
```

```
        </p>
    </card>
</wml>
```



图 6-8　输入界面



图 6-9　显示信息

6.2.2　WML 2.0

WML 2.0 是 WAP 论坛接受 XHTML Basic 标准的产物。WML 2.0 吸收了当前互联网的主要技术标准。WML 2.0 与 W3C 推出的 CSS 简表(CSS Profile)与 XHTML Basic 融合,吸收了 CSS 和 XHTML 的技术特点。这使得 WML 2.0 具有独特的内容:一方面,WML 2.0 中凡是可采用 XHTML Basic 和 CSS 简表表示的内容一概使用 XHTML+CSS 来实现,而原来实现相同功能的 WML 1.x 相关标记取消;另一方面,对于不能使用 XHTML+CSS 设计的内容,仍采用 WML 1.x 的语义来实现,只不过为了与 XHTML+CSS 区别,往往是增加"wml:"前缀以示说明。

WML 用于制作的 WAP 网页,需在 WAP 浏览器中运行。不同于 WML 1.x,WML 2.0 是 XHTML-MP 的超集,具有 XHTML-MP 支持的所有元素和属性,又具有兼容 WML 1.x 的内容。因此,WML 2.0 是由 3 部分构成:XHTML Basic、XHTML 模块元素集和 WML 的扩展元素集,具体内容见表 6-3。

对于继承 XHTML Basic 的元素,具有有两种类型,一方面是为了符合 W3C 制定的标准,具有无 wml 扩展内容的 XHTML Basic 标记;另一方面,为了与 WML 1.x 兼容,在 XHTML Basic 制定的基础上增加了 WML 扩展属性标记,这些标记见表中阴影处。

至于属于 XHTML 模块的标记,是不属于 XHTML Basic 但是 XHTML 中的标记。对于 XHTML 模块的显示模块标记与表单模块标记是和 XHTML 模块与 WML 1.x 兼容的标记,这些标记充分体现向 WML 1.x 兼容,对于 hr 标记是提高性能而加入的标记。

WML 扩展元素集,是兼容 WML 1.x 的主要标记,同时针对新的无线应用而扩展出现的新标记。

表 6-3　WML 2.0 的组成

类别	标　　记	说　　明
XHTML Basic	结构标记 body，head，html，title 文本标记 abbr，acronym，address，blockquote，br，cite，code，dfn，div，em，h1，h2，h3，h4，h5，h6，kbd，p，pre，q，samp，span，strong，var 超链接标记 a 列表标记 dd，dl，dt，li，ul，ol 表单标记 form，input，label，select，option，textarea 表格标记 table，td，th，tr，caption 图像标记 img 对象标记 object，param 链接标记 link 基址标记 base 元信息标记 meta 样式模块标记 style	是基于 XML 语言的标记语言。是 W3C 联盟专门针对受限手持设备而设计的标记语言，是移动版本的 XHTML。WML 2.0 中 XHTML 的这些元素的功能与 XHTML 定义的功能一致。等标记具有 wml 扩展属性
XHTML 模块	显示模块标记 b，big，small，u hr 表单模块标记 fieldset，optgroup	是一些不属于 XHTML Basic 的 XHTML 模块化处理的元素
WML 扩展	导航标记：wml：noop，wml：go，wml：prev，wml：refresh 上下文标记：wml：getvar，wml：setvar，wml：timer，wml：postfield，wml：access，wml：anchor，wml：do 结构标记：wml：card 事件标记：wml：onevent	WML 的扩展元素集

由于 WML 2.0 兼容了 WML 1.x 的特性，又具有当前流行的 XHTML-MP 的所有内容。本章将采用 WAP 论坛的“WAP-238-WML-20010911-a”定义的 WML 2.0 版本制作 WAP 网页。

1. WML 2.0 的文件结构

现在开始学习 WML 2.0。因为 WML 2.0 以 XHTML Basic 为基础，WML 2.0 文件结构与 XHTML 文件结构非常类似。WML 2.0 的文件结构如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD WML 2.0//EN"
"http://www.wapforum.org/dtd/wml20.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:wml="http://www.wapforum.org/2001/wml">
    <head>
       <title><!--定义标题--></title>
    </head>
     <wml:card><!--定义卡片 1--></wml:card>
     …
    <wml:card><!--定义卡片 n--></wml:card>
   </html>
```

或

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD WML 2.0//EN"
"http://www.wapforum.org/dtd/wml20.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:wml="http://www.wapforum.org/2001/wml">
     <head>
        <title><!--定义标题--></title>
     </head>
      <body>
        <!--主体-->
      </body>
    </html>
```

WML 文件结构同 XHTML 文件结构类似。但不同之处在于，WML 2.0 可以采用两种形式，一种采用 XHTML 结构形式，用 body 来定义网页的内容，值得注意的是“body”元素可以定义 WAP 网页内容，但是它不能和“wml：card”同时使用。另一种形式是通过元素“wml：card”取代了 XHTML 中的“body”元素定义 WAP 网页内容，“wml：card”则是定义 WAP 网页的卡片。“wml：card”元素可以出现多次，表示定义多张卡片。这些卡片将文件内容分成不同部分的相关联的片段。

在下列程序清单 6-6 的 WML6-6.wml 文件中，展示了一个用 WML 2.0 编写的 WAP 简单网页。

程序清单 6-6：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD WML 2.0//EN"
    "http://www.wapforum.org/dtd/wml20.dtd">
<!--WML6-6.wml-->
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:wml="http://www.wapforum.org/2001/wml">
    <head><title>第一个 WAP 网页</title></head>
    <wml:card>
        <h1>Welcome to learn WML 2.0</h1>
    </wml:card>
</html>
```

2. WML 2.0 的导航

WML 2.0 兼容 WML 1.x 的处理导航的元素。WML 2.0 实现 WAP 网页导航的标记有“wml：go”、“wml：prev”、“wml：refresh”和“wml：noop”，兼容了 WML 1.x 的同样功能的标记。通过这些标记，可以在 WAP 网页的不同卡片中转换，或执行一定的动作。

3. WML 2.0 的表单制作

WML 2.0 也可以定义表单。通过表单以实现信息的设置，人机的友好交互，以及利用表单发送信息。这些功能与 HTML 网页中表单的功能类似，定义表单的元素与 XHTML 的一致。但是，受到移动设备硬件的限制，表单中的一些元素以及相关属性带有移动设备的特点。另外，WML 2.0 定义的表单控件也不及 HTML 中定义的控件丰富。WML 2.0 只有文本框、文本区、密码框、单选按钮、复选按钮、菜单、发送按钮、重置按钮以及隐藏控件。表 6-4 列出了表单制作的常见元素以及属性。

表 6-4　表单的常见元素及属性

元素	属　　性	说　　明
form	id、class、title、style、action、method 和 enctype	说明定义表单
textarea	id、class、title、style、rows、cols、accesskey、tabindex、wml：emptyok、wml：format、wml：name	定义文本区
input	id、title、style、type、name、size、checked、src、maxlength、tabindex、wml：emptyok、wml：format 和 wml：name	定义输入框
select	id、class、title、style、name、size、multiple、tabindex、wml：iname、wml：ivalue、wml：name 和 wml：value	定义选择列表
option	id、class、title、style、selected、value 和 wml：onpick	定义选项
postfield	id、class、title、name 和 title	定义发送服务器信息

（1）文本控制。WML 2.0 中可以实现文本信息可以通过两种元素：input 和 textarea。input 元素的定义形式同 WML 1.3 的要求。至于 textarea 元素是定义文本区，它必须设置属性 rows 和 cols 两者之一。定义文本区的行数或列数或行数和列数。

此外，input 元素和 textarea 元素有 3 种属性是 XHTML 所没有的，它们是：wml：emptyok、wml：format 和 wml：name。对于其他表单控件出现同样的属性也具有相同的含义。

① wml：emptyok 表示控件中是否接受空内容的输入与否。如果 wml：emptyok 的值为 false，表示不接受空内容的输入；如果 wml：emptyok 取值为 true，则接受空内容的输入。

② wml：format 定义文本输入的格式。WML 2.0 定义一些格式符表示特定的含义。具体见与表 6-2 相同。

③ wml：name 定义具体控件的名称。

（2）选择列表。WML 2.0 继承了 WML 1.x 的选择列表的元素。通常可以用 select 元素和 option 元素组合可以定义菜单。option 定义菜单中的各个选项。这是 WAP 网页菜单定义常见的应用形式。

为了迎合移动手持设备的要求，WML 2.0 中具有 4 种特殊的属性：wml：name、wml：value、wml：iname 和 wml：ivalue。select 的 wml：name 属性的作用是设置控件的名称。wml：value 是指定控件的默认选项的索引值。wml：iname 决定指定默认选项的索引的名称。wml：ivalue 表示如果默认的索引值没有指定，则取 wml：ivalue 变量设定的值。

对于 option 元素中有一个属性是 wml：onpick，它的功能是为指定的 option 设置转向的 URL。

（3）复选控制和单选控制。WAP 网页也需要多选处理和单项选择。特别是移动商务应用中，这些处理大量的存在。WML 2.0 通过设置 input 元素的属性 type 为 checkbox 实现复选框的定义。如果将 input 元素的属性 type 指定为 radio，则定义一个单选项。

（4）提交按钮和重置按钮。提交按钮可以将表单的内容提交，是表单的一项重要的控件。而重置按钮可以将表单的内容清除，让用户重新输入信息。这两种控件也是通过设置 input 元素的 type 属性来实现的。如果 input 元素的 type 属性为 submit，则定义提交按钮。如果 type 属性指定为 reset，则实现重置按钮的定义。

（5）隐藏信息控制。WAP 网页的表单也可以实现隐藏信息传送。要实现这样的功能，需要用 input 元素定义一个隐藏控件，具体语法形式如下：

```
<input type="hidden" />
```

4. WML 2.0 的事件

WML 2.0 向后具有了 WML 1.x 同样的事件。WML 2.0 也定义了 4 种事件：timer、enterbackward、enterforward 和 pick。在具体处理方法上与 WML 1.x 类似。

【例 6-6】 一个向后事件示例。这个例子定义第二、三张卡片用 NEXT 命令进入下一张卡片，用 BACK 返回前一张卡片。所有卡片都可以通过 WAP 浏览器的"BACK"按键返回到第一张卡片。具体代码见程序清单 6-7。

程序清单 6-7：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD WML 2.0//EN"
"http://www.wapforum.org/dtd/wml20.dtd">
<!--WML6-7.wml-->
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:wml="http://www.wapforum.org/2001/wml">
    <head>
        <title>BACKWARD EVENT</title>
    </head>
    <wml:onevent type="enterbackward">                <!--定义文件的向后事件-->
      <wml:go href="#first"/>
    </wml:onevent>
    <wml:card id="first" title="FIRST CARD">
        <wml:do type="enterforward" label="NEXT">
          <wml:go href="#second"/>
        </wml:do>
        <p>The first card.</p>
    </wml:card>
    <wml:card id="second" title="SECOND CARD">
        <wml:do type="enterforward" label="NEXT">
          <wml:go href="#third"/>
        </wml:do>
        <wml:do type="enterbackward" label="BACK">  <!--定义向后事件-->
          <wml:go href="#first"/>
        </wml:do>
        <p>The second card.</p>
    </wml:card>
    <!--定义卡片的向后事件-->
    <wml:card id="third" title="THIRD CARD" onenterbackward="#second">
        <wml:do type="enterforward" label="NEXT">
          <wml:go href="#forth"/>
        </wml:do>
        <wml:do type="enterbackward" label="BACK">  <!--定义向后事件-->
          <wml:go href="#second"/>
        </wml:do>
        <p>The third card.</p>
```

```
        </wml: card>
        <wml: card id="forth" title="FORTH CARD">
            <p>The forth card.</p>
        </wml: card>
</html>
```

5. WML 2.0 的变量

为了让 WAP 网页能实现一定的互动性。WML 2.0 定义了变量。一定值是指变量具有的值。在 WAP 网页运行过程中，变量可以根据不同要求修改值。WML 2.0 中通过 wml：setvar 元素或从表单输入设置一个变量的值。

WML 2.0 还可以通过元素 wml：getvar 来获取一个由属性 name 指定变量名对应变量的值。例："＜wml：getvar name＝"stud_id"＞"表示获取变量 stud_id 的值。

6. 增加样式表

WML 2.0 可以与 WCSS（见 WAP 论坛指定的 WAP-239-WCSS-20011026-a）结合使用，为 WAP 网页增加样式，使得显示内容更加生动。WML 2.0 增加样式表与 XHTML 增加样式的形式类似，有 3 种形式。

（1）使用外部样式。可以用 xml-stylesheet 的形式导入一个外部的样式表，形如：

```
<?xml-stylesheet href="样式表文件名" media="handheld" type="text/css" ?>
```

也可以采用 XHTML 中 head 标记中用 link 标记导入一个外部的样式表，形如：

```
<html>
<head>
   <link href="样式表文件名" type="text/css" rel="stylesheet"/>
</head>
⋮
</html>
```

（2）内部定义样式。可在 WML 2.0 的文件中指定定义针对整个文件的内部样式。与 XHTML 一致，是在 WML 2.0 的文件头部用 style 元素定义，下列程序清单 6-8 显示了一个简单使用内部定义样式的实例，执行结果如图 6-10 所示。

程序清单 6-8：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD WML 2.0//EN"
"http://www.wapforum.org/dtd/wml20.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:wml="http://www.wapforum.org/2001/wml">
    <head>
            <title>BACKWARD EVENT</title>
                        <style type="text/css"><!--定义样式-->
                          p { text-align:  center;
                              text-decoration: underline;
                              font-size: larger;
                            }
                        </style>
    </head>
```

图 6-10　带样式的 WAP 网页

```
    <body>
        <p>Welcome into WML 2.0 Tutorial </p>
    </body>
</html>
```

(3) 内联定义样式。内联定义样式，就是在具体的标记中指定 WCSS 的样式。这与 XHTML 使用内联样式一致。例如：

```
<p style="text-align: center;text-decoration: underline">Welcome into WML 2.0 Tutorial </p>
```

6.2.3 XHTML Mobile Profile

WAP 2.0 的浏览器支持 XHTML Mobile Profile（XHTML 移动描述，简称为 XHTML-MP），XHTML-MP 是 XHTML 1.0 的子集，具有 XHTML 所有的语法要求。同时它又是 XHTML Basic 的超集，它具有 XHTML Basic 的所有元素和属性，并不与 WML 1.x 向后兼容。因此 XHTML MP 失去一些功能：XHTML 不再支持锚点、不再支持显示下划线的 u 标记、XHTML MP 不支持 WML 中的多种事件、XHTML MP 不支持软键盘功能、XHTML MP 不再支持 input 标记中的 format 属性、XHTML MP 不支持脚本以及不支持变量等。

与此同时，XHTML-MP 充分适应移动设备的特点，增加了新的性能，实现了移动设备网页内容与样式的分离，这使得无线 Internet 的标记语言与有线的 Internet 的标记语言走向融合。运用 XHTML MP 开发的网站，无论在受限移动设备还是 PC 中都得到支持。当前，XHTML Mobile Profile 是开发 WAP 应用的主要标记语言。

XHTML MP 由两部分组成 XHTML Basic 内容和新增的 XHTML 模块，见表 6-5。

表 6-5　XHTML MP 的组成

类　别	标　　记	说　　明
XHTML Basic	结构标记 body，head，html，title 文本标记 abbr，acronym，address，blockquote，br，cite，code，dfn，div，em，h1，h2，h3，h4，h5，h6，kbd，p，pre，q，samp，span，strong，var 超链接标记 a 列表标记 dd，dl，dt，li，ul，ol， 表单标记 form，input，label，select，option，textarea 表格标记 table，td，th，tr，caption 图像标记 img 对象标记 object，param 链接标记 link 基址标记 base 元信息标记 meta 样式模块标记 style	是基于 XML 语言的标记语言。是 W3C 联盟专门针对受限手持设备而设计的标记语言，是移动版本的 XHTML

类　别	标　　记	说　　明
XHTML 模块	显示模块标记 b，big，small，hr 表单模块标记 fieldset，optgroup 格外增加的属性有：style 属性，在 li 中增加 value 属性，ol 增加了 start 属性	是一些不属于 XHTML Basic 的 XHTML 模块化处理的元素

用 XHTML MP 开发 WAP 网页，与用 XHTML 开发 HTML 网页的结构类似。只是需要在<! DOCTYPE>中指定 XHTML MP 的 DTD，一个 XHTML MP 文件基本结构如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
        "http://www.wapforum.org/DTD/xhtml-mobile10.dtd" >
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>XHTML Mobile Profile Document</title>
     </head>
    <body>
        <!--主体-->
    </body>
</html>
```

在 XHTML MP 中主体的文本必须出现在其他标记内部，而不能直接嵌入 body 标记中，形如“<body>hello</body>”是错误的表示形式，但是“<body><p>hello</p></body>”则是正确的表示形式。

【例 6-7】 一个简单的 XHTML MP 网页的实例，输出 XHTML MP 的简介，执行结果如图 6-11 所示。

程序清单 6-9：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
        "http://www.wapforum.org/DTD/xhtml-mobile10.dtd" >
<html xmlns="http://www.w3.org/1999/xhtml">
<!--XHTML6-9.xhtml-->
    <head><title>XHTML Mobile Profile Document</title>
    </head>
    <body>
        <h3>了解 XHTML Mobile Profile </h3>
        <p>XHTML Mobile Profile是 XHTML移动描述,是 XHTML的子集。
        </p>
    </body>
</html>
```



图 6-11　输出信息

【例 6-8】 编写一个网页用表格制作菜单，通过它导航到另一个页面，来显示一个表单的实例。菜单页面见程序清单 6-10，显示实例的页面见程序清单 6-11，程序清单 6-12 处理

两个网页的显示的样式，执行结果如图 6-12 和图 6-13 所示。

程序清单 6-10：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
          "http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<!--XHTML6-10.xhtml-->
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>XHTML MP 实例</title>
     </head>
    <body>
        <table>
          <tr><td><img src="1.wbmp"/></td>
              <td><a href="XHTML6-11.xhtml">字体示例</a></td>
          </tr>
          <tr><td><img src="2.wbmp"/></td>
              <td><a href="XHTML6-11.xhtml">表单示例</a></td>
         </tr>
         <tr><td><img src="3.wbmp"/></td>
              <td><a href="XHTML6-11.xhtml">列表示例</a></td>
          </tr>
        </table>
        <hr/><p>陈铁制作 </p>
    </body>
</html>
```

程序清单 6-11：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
          "http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<!--XHTML6-11.xhtml-->
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>XHTML MP 实例</title>
        <link rel="stylesheet" href="style.css" type="text/css"/>
     </head>
    <body>
        <h2>表单实例</h2>
         <form name="frm1">
            文本框<input type="text" size="5"/><br/>
            密码框<input type="password" size="5"/><br/>
            单选按钮<input type="radio"/><br/>
            复选按钮<input type="checkbox"/><br/>
            <select>选择列表
```

```
                <option>选择列表 1</option>
                <option>选择列表 2</option>
                <option>选择列表 3</option>
            </select>
        </form>
    </body>
</html>
```

程序清单 6-12：

```
/*   style.css   */
caption{ font-weight: bold ; font-style: italic ; font-size: larger}
a       {color: blue ; text-decoration: none }
ol      { list-style-type: upper-roman ; list-style-position: inside }
ul      { list-style-type: square }
li      { list-style-position: inherit }
td,th   { border-width: 1px;border-style: solid ; text-align: center }
h2      { color: red }
p       {color: navy }
```



图 6-12　表格运行示例



图 6-13　表单运行实例

6.3　WMLScript

WMLScript 是 WAP 2.0 应用协议的组成部分，是一种为手持移动设备编程的脚本语言。WMLScript 脱胎于欧洲计算机制造商协议会制定的 ECMAScript 脚本语言。它在 ECMAScript 脚本语言的基础上优化和修正而成，认为是 JavaScript 脚本语言的轻量版本。与 JavaScript 作用于 HTML/XHTML 网页一样，WMLScript 和 WML 1.x 结合可以为客户端创建动态、交互等功能的 WAP 网页。WMLScript 的主要作用如下。

（1）实现用户输入的合法检验。

（2）扩展手持移动设备的功能，例如短信息、SIM 卡等热点应用的编程。

（3）快速生成各种信息提示、输入框和对话框等多种形式的人机界面，提供良好的人机交互。

（4）克服化移动服务网络宽带的限制，提供丰富的程序功能。

不同于其他脚本语言，WMLScript 只作用于 WML 语言，并不能作用于其他语言，如 XHTML Basic、XHTML Mobile Profile 等。不同于 JavaScript 脚本嵌入 HTML 网页，WMLScript 脚本是不能嵌入到 WML 文件内，WMLScript 脚本只能独立存在。WML 文件通过超链接导航到 WMLScript 脚本文件，实现脚本的处理。WMLScript 脚本必须单独保存。

WMLScript 脚本有两种形式：文本形式和二进制形式。如果以文本内容形式存在，则保存为文件扩展名为“.wmls”的文件。如果以二进制形式存在，则需要将文本形式的脚本文件编译，编译过后的脚本文件的扩展名为“.wmlsc”。本章介绍 WMLScript 是如何与 WML 结合实现 WAP 网页的制作。

6.3.1　WMLScript 语法基础

WMLScript 参考了 C 语言的语法，与 C 语言具有相类似的语法结构。在本节中，将从 WMLScript 的基本语法规则、变量和数据类型、操作符和表达式、函数和语句来了解 WMLScript。

1. WMLScript 的基本语法规则

与 C 语言一样，WMLScript 的具有一些基本语法规则。

（1）区分大小写。这意味着在 WMLScript 出现的所有变量名、函数名等任何字符串中出现的字母必须区分大小写，例如字符串“Aa”与字符串“aA”是不同的字符串。

（2）WMLScript 与其他语言一样允许注释，通过注释来了解代码。WMLScript 中支持两种形式的注释：单行注释和多行注释。单行注释是用“//注释”定义，只能出现在一行中。多行注释是用“/* 注释 */”来说明，注释内容可以出现在多行。注释不能嵌套定义。

（3）WMLScript 支持程序代码间空格、换行和制表符忽略。但是空格、换行和制表符出现在字符串内，则按照这些字符的原来含义处理。

（4）WMLScript 定义了特殊含义的保留字。这些保留字见表 6-6。保留字不能作为其他变量名、函数名等出现。

表 6-6　WMLScript 的保留字

access	agent	http	break	continue	isvalid	use
meta	div	div=	name	path	domain	for
else	return	typeof	equiv	extern	url	function
user	var	header	while			

2. WMLScript 的变量和数据类型

WMLScript 中可以定义对应“变量名-值”的变量。变量只能出现在两种情况。一种是

在函数中，用 var 关键字定义。形如："var x=23;"表示定义一个变量，值为整数 23。另一种是传递而来的变量。可以是从 WML 文件传递而来，也可以是通过 WMLScript 脚本的其他函数传递而来的变量。在后面的例子中可以看到这种应用。

尽管变量不像 C 语言一样，在定义时用明确的关键字来确定变量的类型。实际上，WMLScript 的变量也具有一定数据类型。在 WMLScript 中具有 5 种基本数据类型：整数、浮点数、字符串、布尔类型和 invalid。这 5 种变量的取值范围见表 6-7。变量的数据类型取决于具体值的数据类型。例：

```
var x=44;            //表示 x 是整数；
var ipe=invalid;     //表示变量的值为无效；
```

表 6-7　WMLScript 的基本数据类型的取值范围

数据类型	取 值 范 围
整数	−2147483648～2147483647
浮点数	$1.17549435\times10^{-38}\sim3.40282347\times10^{38}$
字符串	由字母、数字或其他字符构成，字符串包含在用一对""包含起来
布尔	true、false
INVALID	invalid，表示无效数据

3. 操作符和表达式

WMLScript 语言提供了 WML 所不具有的运算功能。WMLScript 的运算功能可以弥补 WML 在制作 WAP 网页中的不足。WMLScript 提供了算术运算、逻辑运算、位运算、赋值运算、比较运算、字符串运算、条件运算和类型运算。在下列表 6-8 中罗列出执行这些运算的操作符。WMLScript 的这些运算操作符、变量和常量结合可以形成一个表达式，表示一个具有特定含义的式子。例如：(x>y)? x+2：y+3 就是一个表示条件运算的表达式。又如：(x+y)&&(z+4)也是一个多种运算结合的表达式。

表 6-8　WMLScript 的运算操作符

类别	操作符	含　　义
算术运算	+	加法运算
	−	减法运算
	*	乘法运算
	/	除法运算
	div	整除运算
	%	求余运算
	++	自增运算
	−−	自减运算

<table>
<tr><th>类别</th><th>操作符</th><th>含　　义</th></tr>
<tr><td rowspan="7">位运算</td><td><<</td><td>左移位运算</td></tr>
<tr><td>>></td><td>右移位运算，最高的符号位不变</td></tr>
<tr><td>>>></td><td>右移位运算，最高位补 0</td></tr>
<tr><td>&</td><td>位与运算</td></tr>
<tr><td>|</td><td>或运算</td></tr>
<tr><td>^</td><td>异或运算</td></tr>
<tr><td>~</td><td>非运算</td></tr>
<tr><td rowspan="3">逻辑运算</td><td>&&</td><td>与运算</td></tr>
<tr><td>||</td><td>或运算</td></tr>
<tr><td>!</td><td>非运算</td></tr>
<tr><td>赋值运算</td><td>=</td><td>赋值运算，=可以和其他二值运算操作符结合成执行不同类型的赋值操作，有+=、-=、* =、/ * 、div=、<<=、>>=、>>>=、&=、|=、^=、~=、&&=、||=、! =</td></tr>
<tr><td>字符串运算</td><td>+</td><td>字符串连接运算</td></tr>
<tr><td rowspan="6">比较运算</td><td><</td><td>小于运算</td></tr>
<tr><td><=</td><td>小于等于运算</td></tr>
<tr><td>==</td><td>等于运算</td></tr>
<tr><td>></td><td>大于运算</td></tr>
<tr><td>>=</td><td>大于等于运算</td></tr>
<tr><td>! =</td><td>不等于运算</td></tr>
<tr><td>条件运算</td><td>?:</td><td>条件运算，例:? (2>3)3:1 结果为 1</td></tr>
<tr><td>类型运算</td><td>typeof</td><td>类型运算，获取基本数据类型对应的代码</td></tr>
<tr><td>isvalid 运算</td><td>isvalid</td><td>判断值是否为无效，为无效返回 false，为有效返回 true，例如 isvalid(3/0)返回为 false</td></tr>
</table>

4. 语句

与 C 语言一样，WMLScript 的关键字与表达式结合形成语句。与表达式不同的是，语句必须要用分号“;”作为结尾。语句可以有空语句，即语句中只有一个分号，而没有其他任何代码。空语句为程序员的后续编写程序提供了位置。此外，WMLScript 提供了条件控制的相关语句。这些语句为灵活处理数据提供了方便。

(1) if 语句。if 语句提供了条件判断。它有两种形式：

形式 1：

```
if (表达式 1)
   表达式 2;          //表示如果表达式为真,结果为表达式 2,否则为 invalid;
```

形式 2：

```
if(表达式 1)
  表达式 2;
else
表达式 3;            //表示如果表达式为真,结果为表达式 2,否则为表达式 3;
```

(2) while 语句。while 语句可以执行循环。对于某些重复性的操作可以通过 while 语句来实现。具体的语法定义形式如下:

```
while(表达式)
   语句;
```

表示的含义是如果表达式为真,则执行循环体中的语句。当表达式的值为假或 invalid 无效时,则执行 while 语句的后续语句。

(3) for 语句。for 语句是另一种循环的语句。for 语句的语法形式有两种形式:

```
for(表达式 1;表达式 2;表达式 3)
    语句
```

和

```
for(var 变量声明列表;表达式 2;表达式 3)
    语句
```

for 语句的组成用";"分成 3 个部分:第一部分一般处理变量的初始化,第二部分表示条件判断,第三部分表示执行完循环体后修改循环条件。例如:

```
for (var i=0; i<100; i++) {//执行 100 次加法运算。
    result +=i;
};
```

(4) continue 语句和 break 语句。WMLScript 中提供了 continue 语句和 break 语句。这两个语句都和循环相关联。continue 表示结束本次循环,而 break 语句表示跳出循环语句,结束循环操作。

(5) return 语句。return 语句是和函数紧密联系的一个语句。语法形式如下:

```
return [表达式];
```

"表达式"用"[]"包括起来,表示它是一个可选项。如果没有表达式,则它的作用是从函数中返回,并带有一个空字符串。如果有表达式,表示从函数返回,并带有一个由表达式决定的返回值。

5. 函数

函数是 WMLScript 的中重要组成部分,通过函数定义特定功能。函数声明的语法定义形式如下:

```
[extern] function 函数名 ([参数名表])
{  函数体  }
```

函数的声明中,有几点需要注意。

(1) 关键字 extern 是可选项。如果在声明时,使用 extern,表示该函数是外部函数,可以被其他文件所使用;如果默认为 extern,表示函数仅仅在定义文件内部有效,是一个内部函数;

(2) 函数名必须符合 WMLScript 的基本语法规则。此外,函数名不能相同。

(3) 函数有返回值,如果默认为 return 语句,函数的返回值为空字符串。

(4) 函数参数可以是 0 个或多个,通过参数名表定义。函数的参数可以传递数据,在函数调用时,函数的参数个数和参数类型必须和声明的内容一致。

(5) 调用内部函数时仅仅需要给出函数名和实参即可。如果在 WMLScript 脚本调用其他文件的外部函数需要通过预编译语句,形如:"use url 外部文件名 "具体的 URL""来指定外部函数的位置。然后,才能调用外部函数。调用时用"外部文件名 # 外部函数(实参)"。

6.3.2 WMLScript 常用库

WMLScript 的库是将预先定义的函数组成一个实体。WMLScript 定义的标准库有 Dialogs、Float、String、Lang、URL 和 WMLBrowser 这 6 种类型,这些库中包括的函数见表 6-9。调用标准库的函数可以采用形如"标准库名.函数名(实参表)"来实现。

表 6-9 WMLScript 的标准库

库	常见函数	含义
Dialogs	alert(message)	显示信息
	confirm(message,ok,cancel)	定义用户选择信息框
	prompt(message,defaultInput)	定义弹出信息框
Float	ceil(value)	返回不小于数本身的整数值
	floor(value)	返回不大于本身的整数值
	int(value)	返回整数部分
	maxFloat()	最大的浮点数
	minFloat()	最小的浮点数
	pow(x,y)	返回 x 的 y 次方
	round(value)	返回最接近 value 的整数
	sqrt(value)	返回 value 平方根
Lang	parseFloat(value)	返回由字符串 value 所定义的浮点数值
	parseInt(value)	返回由参数字符串所定义的整数值
	random(value)	返回一个随机正整数
	isFloat(value)	判断 value 是否是浮点数
	inInt(value)	判断 value 是否是整数

库	常见函数	含义
String	removeAt(s,index,sep)	返回删除后的字符串
	subString(s,index,length)	返回字符串的子串
	length(string)	返回字符串的长度
	toString(value)	返回 value 的字符串形式
	trim(string)	删除字符串前导空格
	replace(string,oldsub,newsub)	返回用 newsub 替换 oldsub 后的新串
	replaceAt(s,element,index,sep)	返回指定位置替换后的新串
	compare(s1,s2)	返回比较 s1 和 s2 的值
	format(format,value)	返回 value 按照指定格式的字符串
URL	getHost(url)	返回 URL 主机名
	getParameters(url)	返回 URL 的参数表
	getPath(url)	返回 URL 的路径
	getPort(url)	返回 URL 的端口
WMLBrowser	getVar(name)	返回指定变量名的值
	go(url)	指示浏览器装载指定 url 的网页
	prev()	指示浏览器返回前一张卡
	refresh()	指示浏览器刷新当前的卡
	setVar(name,value)	指示浏览器修改指定变量的值
	getCurrentCard()	返回当前卡的 URL
	nextContext()	清除浏览器的上下文

【例 6-9】 设计 WAP 页面,实现验证用户登录信息是否正确。假设用户名为“chen”,密码为“123456”。如果用户输入信息正确,浏览器显示欢迎信息,并在 5s 后转向到 tutorial. wml 页面。如果登录信息错误,则显示登录失败信息,并在 5s 导航到登录界面。具体代码见程序清单 6-13(定义 WAP 页面)和程序清单 6-14(定义 WMLScript 脚本验证登录信息)运行结果如图 6-14 和图 6-15 所示。

程序清单 6-13:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
     "http://www.wapforum.org/DTD/wml13.dtd">
<!--WML6-13.wml-->
<wml>
  <card id="log" title="登录">
    <p align="center">
      用户名<br/>
```

```
        <input type="text" name="username"/><br/>
        密码<br/>
      <input type="password" name="password"/><br/>
      <do  type="accept" label="登录">
        <go href="WMLS6-14.wmlsc#checkUsername('$ (username)','$ (password)')"/>
        </do>
    </p>
  </card>
  <card id="Welcome" title="欢迎"><!--输出欢迎界面-->
        <onevent type="ontimer"><!--5秒后导航到 tutorial.wml-->
            <go href="tutorial.wml"/>
        </onevent>
        <timer value="50"/>
         <p align="center">
         <big><b>欢迎$ (username)来到 WMLScript 教程.</b></big>
    </p>
  </card>
<card id="Error" title="错误"><!--输出错误界面-->
    <onevent type="ontimer"><!--5秒后返回第一张卡片-->
            <go href="#log"/>
        </onevent>
        <timer value="50"/>
    <p align="center">
        <big><b>$ (username)请重新输入正确登录信息!!!</b></big>
    </p>
</card>
</wml>
```




图 6-14　登录界面



图 6-15　登录失败

程序清单 6-14：

```
//WMLS6-14.wmls
extern function checkUsername(username,keyword) {
if(username=="chen"&&keyword=="123456")
    WMLBrowser.go("WML6-13.wml#Welcome");
   else
    WMLBrowser.go("WML6-13.wml#Error");
}
```

小　　结

无线 Internet 已经成为目前的一个应用热点。本章初步介绍了 WAP 2.0 的发展以及主要特点，并着重介绍了 WAP 2.0 的主要的标记语言 WML 1.3、WML 2.0 和 XHTML MP。对这些标记语言之间的关系，做了深入的说明。XHTML MP 是 XHTML Basic 的子集，将无线 Internet 与有线 Internet 世界建立桥梁，是当前的主要 WAP 网站开发语言。WML 2.0 扩展 XHTM MP，一方面向 WML 1.x 兼容，同时又符合 XHTML 标准的要求。此外，本章还介绍了 WMLScript。着重说明 WMLScript 的语法规则，以及常用的标准库。通过对它们的介绍，可以初探应用于受限手持移动设备的 WAP 网站的制作。

练　习　6

1. 填空

(1) WML 是________语言，而 WMLScript 代表________语言，XHTML MP 表示________。

(2) WMLScript 的函数类别分成________和________。

(3) WML 实现的导航的标记有________、________、________和________四种。

(4) 获取 WML 的变量的可以通过________实现。

2. 操作题

(1) 下列是 WML 2.0 的程序段，请指出错误，并改正。

```
 ⋮
<wml:card id="first" title="Menu">
<form action="">
        <select><!--定义查询菜单 -->
          <option wml:onenterforward="#second">按站站查询</option>
          <option wml:onenterforward="#third">按车次查询</option>
        </select>
      </form>
    </wml:card>
   ⋮
```

(2) 请将下列的程序段空格部分补充完整，要求实现 10s 后，导航到第二张卡片。

```
⋮
    <head>
        <title>timer Event Demo</title>
    </head>
    <wml:card id="first" title="Menu" ________>
        ________
        <p>Welcome to the web site </p>
    </wml:card>
    <wml:card id="second" title="none">
        <p><img src="image/p1.wbmp" alt="icon of the web site"/></p>
    </wml:card>
</html>
```

3. 实验题

(1) 分别用 WML 1.3、WML 2.0 以及 XHTML 开发一个火车时刻表查询界面。

(2) 设计并实现简易计算器,能实现加、减、乘和除的运算。

第 7 章　JSP 开发的 Java 语言基础

7.1　Java 简介

1995 年，美国 Sun Microsystems 公司正式向 IT 业界推出了 Java 语言，该语言具有安全、跨平台、面向对象、简单、适用于网络等显著特点，当时以 Web 为主要形式的互联网正在迅猛发展，Java 语言的出现迅速引起所有程序员和软件公司的极大关注，程序员们纷纷尝试用 Java 语言编写网络应用程序，并利用网络把程序发布到世界各地进行运行。包括 IBM、Oracle、微软、Netscape、Apple 和 SGI 等大公司纷纷与 Sun Microsystems 公司签订合同，授权使用 Java 平台技术。

7.1.1　Java 语言特点

1. 简单、面向对象

Java 的简单首先体现在精简的系统上，力图用最小的系统实现足够多的功能；对硬件的要求不高，在小型的计算机上便可以良好地运行。和所有的新一代的程序设计语言一样，Java 也采用了面向对象技术并更加彻底，所有的 Java 程序和 Applet 程序均是对象，封装性实现了模块化和信息隐藏，继承性实现了代码的复用，用户可以建立自己的类库。而且 Java 采用的是相对简单的面向对象技术，去掉了运算符重载、多继承的复杂概念，而采用了单一继承、类强制转换、多线程、引用（非指针）等方式。无用内存自动回收机制也使得程序员不必费心管理内存，使程序设计更加简单，同时大大减少了出错的可能。

2. 鲁棒并且安全

Java 语言在编译及运行程序时，都要进行严格的检查。作为一种强类型语言，Java 在编译和连接时都进行大量的类型检查，防止不匹配问题的发生。如果引用一个非法类型、或执行一个非法类型操作，Java 将在解释时指出该错误。在 Java 程序中不能采用地址计算的方法通过指针访问内存单元，大大减少了错误发生的可能性；而且 Java 的数组并非用指针实现，这样就可以在检查中避免数组越界的发生。无用内存自动回收机制也增加了 Java 的鲁棒性。

作为面向网络语言，Java 必须提供足够的安全保障，并且要防止病毒的侵袭。Java 在运行应用程序时，严格检查其访问数据的权限，比如不允许网络上的应用程序修改本地的数据。下载到用户计算机中的字节代码在其被执行前要经过一个核实过程，一旦字节代码被核实，便由 Java 解释器来执行，该解释器通过阻止对内存的直接访问来进一步提高 Java 的安全性。同时 Java 极高的鲁棒性也增强了 Java 的安全性。

3. 结构中立并且可以移植

网络上充满了各种不同类型的计算机和操作系统，为使 Java 程序能在网络的任何地方运行，Java 编译器编译生成了与体系结构无关的字节码结构文件格式。任何种类的计算

机，只有在其处理器和操作系统上有 Java 运行时环境，字节码文件就可以在该计算机上运行。即使是在单一系统的计算机上，结构中立也有非常大的作用。随着处理器结构的不断发展变化，程序员不得不编写各种版本的程序以在不同的处理器上运行，这使得开发出能够在所有平台上工作的软件集合是不可能的。而使用 Java 将使同一版本的应用程序可以运行在所有的平台上。

体系结构的中立也使得 Java 系统具有可移植性。Java 运行时系统可以移植到不同的处理器和操作系统上，Java 的编译器是由 Java 语言实现的，解释器是由 Java 语言和标准 C 语言实现的，因此可以较为方便地进行移植工作。Java 的可移植性原理如图 7-1。

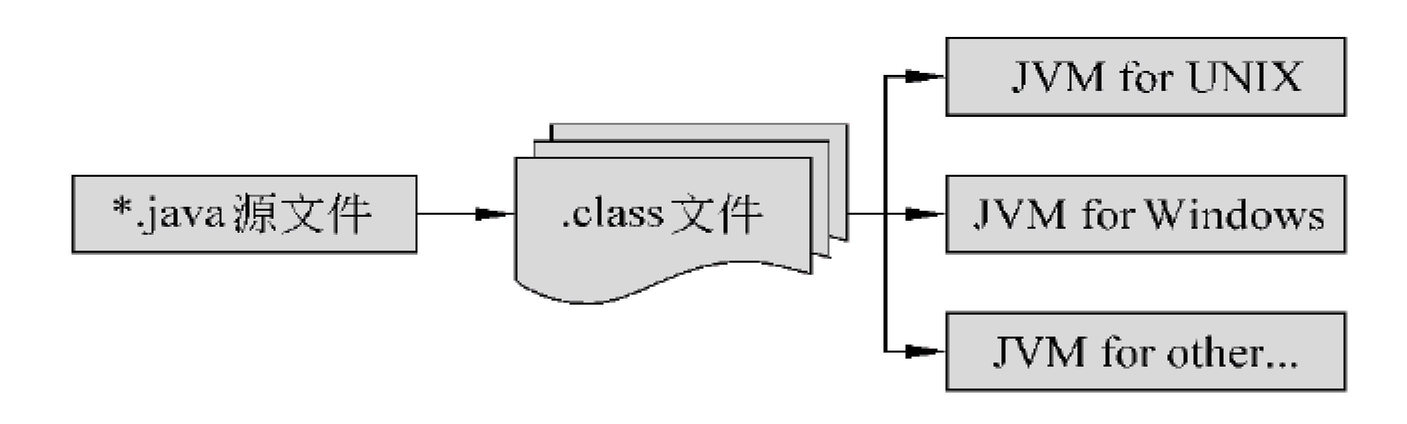


图 7-1　Java 跨平台

4. 高性能

虽然 Java 是解释执行的，但它仍然具有非常高的性能，在一些特定的 CPU 上，Java 字节码可以快速地转换成为机器码进行执行。而且 Java 字节码格式的设计就是针对机器码的转换，实际转换时相当简便，自动的寄存器分配与编译器对字节码的一些优化可使之生成高质量的代码。随着 Java 虚拟机的改进和“即时编译”(just in time)技术的出现使得 Java 的执行速度有了更大的提高。

5. 解释执行、多线程并且是动态的

为易于实现跨平台性，Java 设计成为解释执行，字节码本身包含了许多编译时生成的信息，使连接过程更加简单。而多线程使应用程序可以同时进行不同的操作，处理不同的事件。在多线程机制中，不同的线程处理不同的任务，互不干涉，不会由于某一任务处于等待状态而影响了其他任务的执行，这样就可以容易地实现网络上的实时交互操作。Java 在执行过程中，可以动态地加载各种类库，这一特点使之非常适合于网络运行，同时也非常有利于软件的开发，即使是更新类库也不必重新编译使用这一类库的应用程序。

7.1.2　Java 和 JavaScript 的区别

初看起来，两者似乎有很近的血缘关系，但是事实上 JavaScript 与 Java 是由不同的公司开发的不同产品。Java 是 Sun 公司推出的新一代面向对象的程序设计语言；而 JavaScript 是 Netscape 公司的产品，其目的是为了扩展 Netscape Navigator 功能，而开发的一种可以嵌入 Web 页面中的基于对象和事件驱动的解释性语言，它的前身是 Live Script，而 Java 的前身是 Oak 语言。两者的主要区别如下。

1. 面向对象与基于对象

Java 是一种真正的纯面向对象编程语言，在 Java 中，一切都是对象；JavaScript 是一种脚本语言，由于它本身提供了非常丰富的内部对象供程序员使用，因而它是基于对象的语言。

2. 开发和运行环境的不同

若利用 Java 编写程序并使之运行，必须事先在系统内安装相应版本的 JDK 和 JVM，保证代码能够得到编译和运行的环境；而编写 JavaScript 则相对简单，只需使用某种 HTML 文档编辑器甚至某种字符编辑器，如 Notepad，然后打开浏览器即可运行。

3. 解释和编译

两种语言在其浏览器中所执行的方式不一样。Java 的源代码在传递到客户端执行之前，必须经过编译，因而客户端上必须具有相应平台上的仿真器或解释器，它可以通过编译器或解释器实现独立于某个特定的平台编译代码的束缚。JavaScript 是一种解释性编程语言，其源代码在发往客户端执行之前不需经过编译，而是将文本格式的字符代码发送给客户编由浏览器解释执行。

4. 变量类型不一样

两种语言所采取的变量是不一样的。Java 采用强类型变量检查，即所有变量在编译之前必须声明。如：

```
Integer x; String y; x=1234; y="4321";
```

其中 x=1234 说明是一个整数，y="4321"说明是一个字符串。

JavaScript 中变量声明，采用弱类型。即变量在使用前不需声明，而是解释器在运行时检查其数据类型，如：

```
x=1234;
y="4321";
```

前者说明 x 为数值型变量，而后者说明 y 为字符型变量。

5. 代码格式不一样

Java 是一种与 HTML 无关的格式，必须通过像 HTML 中引用外媒体那样进行装载，其代码以字节代码的形式保存在独立的文档中。

JavaScript 的代码是一种文本字符格式，可以直接嵌入 HTML 文档中，并且可动态装载。

6. 嵌入方式不一样

在 HTML 文档中，两种编程语言的标识不同，JavaScript 使用<script>…</script>来标识，而 Java 使用<%... %>来标识。

综上所述，可以发现，JavaScript 与 Java 虽然都可以应用于网页设计，但它们有太大的区别，是两种不同语言。

7.2 Java 的基本语法

7.2.1 数据类型

本节将介绍 Java 语言的数据类型。同其他所有的高级编程语言一样，Java 支持多种数据类型，它们可以用来声明变量、创建数组以及其他更复杂的数据结构。Java 的数据类型如图 7-2 所示。

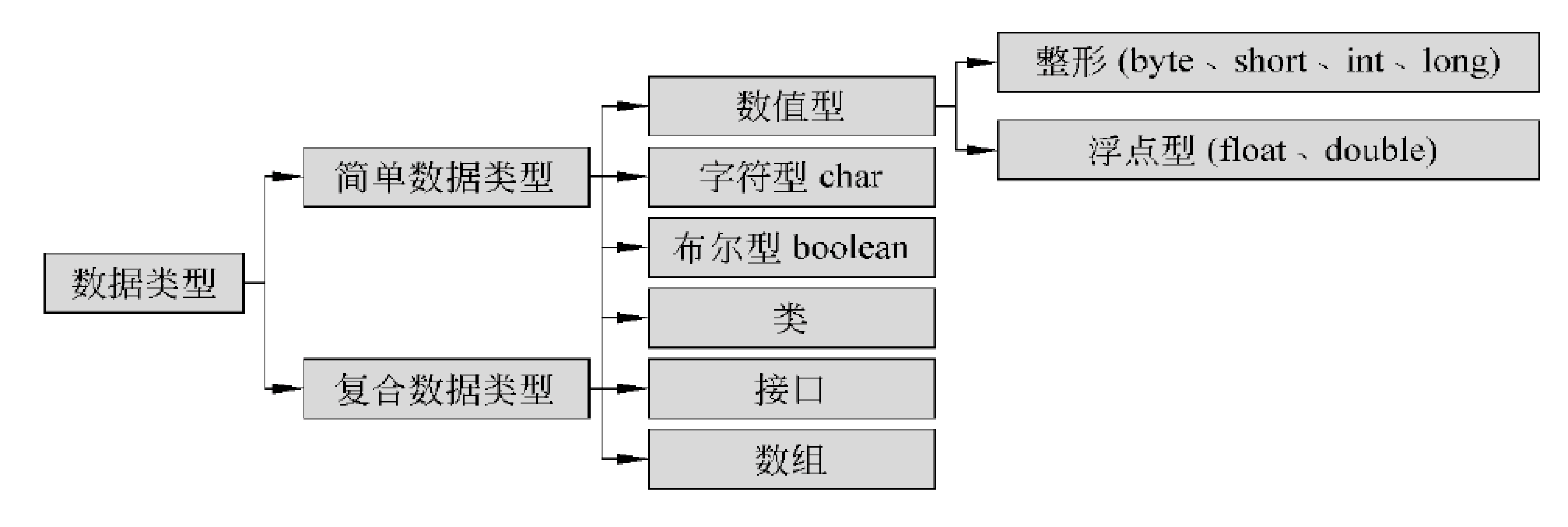


图 7-2　Java 数据类型

本节先来讨论简单数据类型,复合数据类型留到后面的章节再作介绍。

1. 整型

Java 各整数类型有固定的表数范围各字段长度见表 7-1,其不受具体操作系统的影响,以保证 Java 程序的可移植性,这点与 C 语言是不同的。

Java 语言整型常量有 3 种表示形式。

十进制整数:如 123,−456,0

八进制整数:以 0 开头,如 0123 表示十进制数 83,−011 表示十进制数−9。

十六进制整数:以 0x 或 0X 开头,如 0x123 表示十进制数 291,−0X12 表示十进制数−18。

Java 默认的整型常量为 int 类型,声明 long 型常量需要在后面加上'l'或'L',

2. 字符类型(char)

字符常量是用单引号括起来的一个字符,如'a','A';

字符型变量的声明是在其前面加关键字 char,它采用 Unicode 编码(全球统一编码),每个字符占两个字节,可用十六进制编码表示,它在机器中占 16 位,其范围为 0～65535。字符型变量的定义如:

```
char c= 'a';        //指定变量 c 为 char 型,且赋初值为 'a'
char c= '\n';       //换行符
```

3. 布尔类型(boolean)

布尔型数据只有两个值 true 和 false,不可以用 0 或非 0 的整数替代 true 和 false,这点与 C 语言不同。布尔类型变量适于做逻辑运算,一般用于程序流程控制。布尔型变量的定义如:

```
boolean b= true;
```

4. 浮点型(实型)数据

与整型类型类似,Java 规定浮点型数据有固定的表数范围和字段长度,见表 7-1,同样也不受平台影响。

Java 浮点型常量有两种表示形式:

十进制数形式:由数字和小数点组成,且必须有小数点,如 0.123,1.23,123.0

科学计数法形式:如:123e3 或 123E3,其中 e 或 E 之前必须有数字,且 e 或 E 后面的

指数必须为整数。

Java 默认的浮点型为 double 型，如要声明一个 float 型常量，则需要在数字后面显式的加上'f'或'F'。

表 7-1 简单数据类型的表数范围和字段长度

类型		占用存储空间/B	表数范围
整型	byte	1	$-128\sim127$
	short	2	$-2^{15}\sim2^{15}-1$
	int	4	$-2^{31}\sim2^{31}-1$
	long	8	$-2^{63}\sim2^{63}-1$
字符型	char	2	$0\sim65535$
浮点型	float	4	$-3.403\times10^{38}\sim3.403\times10^{38}$
	double	8	$-1.798\times10^{308}\sim1.798\times10^{308}$
布尔型	boolean		true/false

在编程时，经常会碰到数据类型的转换问题，Java 语言规定，boolean 类型不可以转换为其他的数据类型，整型、字符型和浮点型数据在混合运算时可以相互转换，但要遵循以下几个原则。

（1）容量小的类型自动转换为容量大的数据类型，即按以下顺序可自动转换：

```
byte,short,char -> int -> long -> float -> double
```

byte，short，char 之间不会直接互相转换，而是首先转换为 int 类型再作相应转换。

（2）容量大的数据类型转换为容量小的数据类型时，要加上强制转换符，但可能造成精度降低或溢出，使用时需要格外注意。

（3）有多种类型的数据混合运算时，系统首先自动地将所有数据类型转换成容量最大的那一种数据类型，然后再进行计算。

7.2.2 数组

在数学中，常用向量（Vector）来表示一些相关的数据，在 Java 程序设计中，则可以用数组来表示向量的概念。数组是多个数据项的有序集合，其中的每个数据项称为数组的元素。在 Java 程序中，数组具有下列特点。

（1）同一数组中的所有元素均属于相同的数据类型，该数据类型称为数组的基本类型。

（2）数组一经创建，其元素个数就保持不变，这个长度称为数组的长度。

（3）数组中的每一个元素均能借助于下标（index）来访问。

（4）数组元素的类型既可以是简单数据类型（如 int，float 等），也可以是复合类型（如 String，Object，甚至数组类型），元素类型如果属于简单数据类型则每个元素都为一个值，如果属于复合类型则每个元素是一个对象。

1. 一维数组

（1）一维数组的声明。在 Java 语言中，一维数组按如下方式声明：

```
type[] arrayName;
```

或者

```
type arrayName[];
```

两种声明格式都是可行的。其中，type 表示数组元素的类型，可以是 8 种简单数据类型之一，也可以是复合数据类型或用户自定义类型；“[ ]”用于表明定义的是一维数组；arrayName 就是所要声明的数组名字，或者说是声明了一个名为 arrayName 指向“type”类型数组的引用。注意，Java 语言中声明数组时不能指定其长度（数组中元素的个数），例如：

```
int a[5];          //非法
```

声明数组时，并没有为其中的元素分配存储空间，而要等到创建数组的时候才真正为它们分配存储空间。在 Java 中，数组的创建必须使用“new”操作符，创建一维数组的语法格式如下：

```
arrayName=new elementType[ARRAY_SIZE];
```

其中，arrayName 为已经声明的一维数组名；elementType 为数组基类型，它必须是声明 arrayName 时指定的数组基类型或其子类型；ARRAY_SIZE 为整数类型常量，它指定了数组的长度。必须注意，数组一旦创建，其长度就不能改变。

（2）一维数组元素的引用。创建了一维数组之后，如果想对数组的元素赋值，或者读取数组元素的值，则可以按如下语法格式来访问一维数组元素：

```
arrayName[index];
```

其中 arrayName 是数组名，index 是一个整型数值，用于表示要访问的元素下标，例如：intArray[6]访问的是 intArray 中下标为 6 的元素。需要注意的是在 Java 语言中数组元素的下标是从 0 开始，直到数组长度减 1。每个数组都有一个属性 length 指明它的长度，例如：intArray.length 指明数组 intArray 的长度。

（3）一维数组的初始化。定义和创建完数组后，如果没有对数组元素进行初始化，则数组元素将采用对应于该元素类型的默认值，如 int 数组中元素的默认值是 0，对象数组中元素的默认值是 null 等。对于一维数组，可以采用如下方法进行初始化：

动态初始化：数组定义与数组元素分配空间和赋值的操作分开进行，例如程序清单 7-1：

程序清单 7-1：

```
//OneDimensionDynamicInit.java
public class OneDimensionDynamicInit {
    public static void main(String[] args) {
        int a[];
        a=new int[3];
        a[0]=3; a[1]=9; a[9]=8;
        Date days1[], days2[];
        days1=new Date[3];
        days1[0]=new Date(2007, 5, 4);      //为 days[0]单元开辟空间
```

```
            days1[1]=new Date(2008, 3, 2);      //为 days[1]单元开辟空间
            days1[2]=new Date(2008, 10, 1);     //为 days[2]开辟空间,对应与图 7-5 (b)
            days2=new Date[30];
            for(int i=0; i<30; i++)             //通过循环中初始化
                days2[++i]=new Date(2008, 1, ++i);
            }
    }
    class Date {
        int year, month, day;
        Date(int y, int m, int d) {
            this.year=y;
            this.month=m;
            this.day=d;
        }
    }
```

静态初始化:在定义数组的同时就为数组元素分配空间并赋值,例如:

```
int a[]={3, 9, 8, 5};
Date days[]={new Date(2007, 5, 4),
             new Date(2008, 3, 2)
             };
```

2. 多维数组

多维数组就是数组的数组,数组元素的数据类型还是数组类型。在多维数组中访问一个元素要用到不只一个数组下标。如果一个数组的基类型(数组元素的数据类型)是一维数组,则该数组称为二维数组,以此类推。

(1) 多维数组的定义

在 Java 中,定义二维数组的语法格式如下:

```
elementType[][]    arrayName;
```

同样的,定义三维数组的语法格式如下:

```
elementType[][][]    arrayName;
```

其中,elementType 为数组元素的类型,可以是基本数据类型,也可以是复合数据类型;arrayName 为数组名字。

与一维数组一样,在 Java 中创建多维数组也要用到 new 运算符,创建二维数组的语法格式如下:

```
arrayName=new elementType [DIMENSION_1_SIZE][DIMENSION_2_SIZE];
```

其中,arrayName 为二维数组的名字;elementType 为数组元素的类型,必须是定义 arrayName 时指定的数组元素类型或其子类型,DIMENSION_1_SIZE 和 DIMENSION_2_SIZE 分别指明数组各维的长度。

(2) 多维数组元素的引用。引用多维数组中的元素时要用到多个数组下标,一般的语

法格式如下：

```
arrayName [dimension_1_index] [dimension_2_index]… [dimension_n_index];
```

其中 arrayName 为数组名字；n 为数组的维数；dimension_1_index～dimension_n_index 均为整型变量，代表所要访问的元素在各维中的下标。

（3）多维数组的初始化。与一维数组类似，多维数组既可以用循环遍历的方式对每个元素进行初始化，也可以用初始化列表进行初始化。下面分别介绍多维数组的动态和静态初始化方法。

动态初始化

```
Date days1[][]=new Date[3][2];
Date days2[][]=new Date[3][];
days2[0]=new Date[3];
days2[1]=new Date[5];
days2[2]=new Date[6];
```

静态初始化

```
int a[][]={{1, 2}, {3, 4}, {3, 4, 5}};
int b[3][2]={{1, 2}, {3, 4}, {4, 5}};          //非法
```

7.2.3 常用运算

在 Java 语言中，运算符可以划分为四大类：算术运算符、关系运算符、位运算符以及逻辑运算符。此外还定义了一些附加运算符，用于某些特殊情况的处理。

1. 算术运算

常见的算术运算符有：＋，－，＊，/，%，＋＋，－－。它们的主要作用对象是整型和浮点型数据。这里运算符可以分为双目运算符和单目运算符。“目”就是操作数的个数，双目运算符是连接两个操作数的运算符，单目运算符只需要一个操作数。

2. 关系运算

关系运算符用于判断两个操作数之间的关系，运算结果为一个布尔值(true 或 false)。Java 定义了 6 种关系运算符。分别如表 7-2 所示：

表 7-2 Java 运算符

运 算 符	名 称	示 例
＜	小于	a＜b
＞	大于	a＞b
＜＝	小于等于	a＜＝b
＞＝	大于等于	a＞＝b
＝ ＝	等于	a＝ ＝b
！ ＝	不等于	a！ ＝b

3. 逻辑运算

逻辑运算分为逻辑与、逻辑或、逻辑非、逻辑异或运算等。具体的运算规则如表 7-3 所示：

表 7-3 Java 逻辑运算概览

运算数 1	运算数 2	逻辑非	逻辑与	逻辑或	逻辑异或	短路与	短路或
a	b	! a	a&b	a|b	a^b	a&&b	a||b
true	true	false	true	true	false	true	true
true	false	false	false	true	true	false	true
false	true	true	false	true	true	false	true
false	false	true	false	false	false	false	false

4. 位运算

Java 中的位运算就是把二进制运算扩展到两个操作数的每一位上进行就可以了，具体情况与二进制的位运算是一样的，读者如果不清楚可以参照其他参考书。

7.2.4 控制语句

1. if 语句

if 语句常常用于条件判断的选择结构中，可以嵌套。它的两种不同的语句形式如图 7-3 和图 7-4 所示。

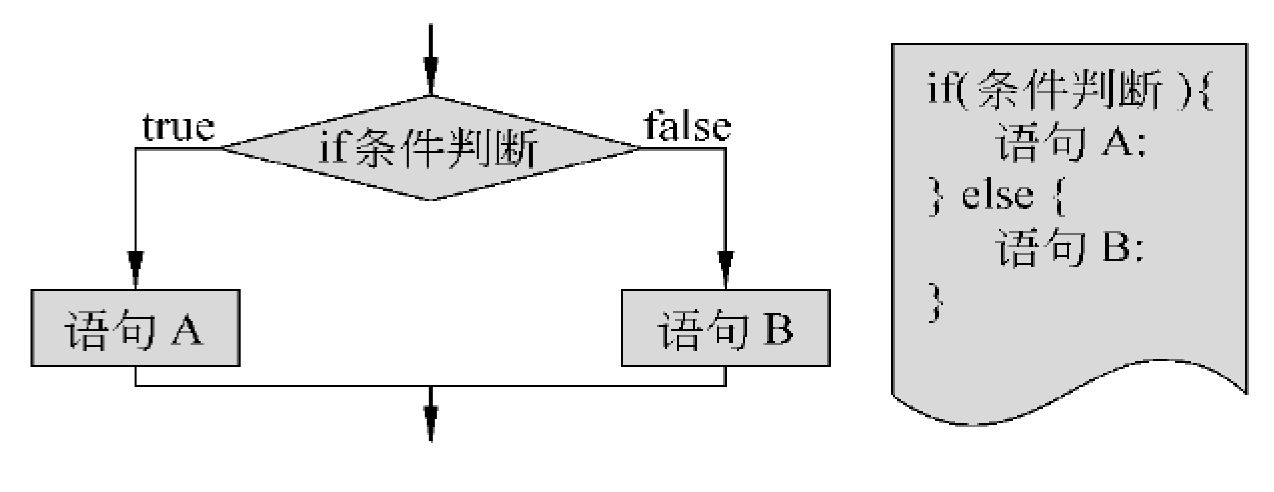


图 7-3 if 语句形式 1

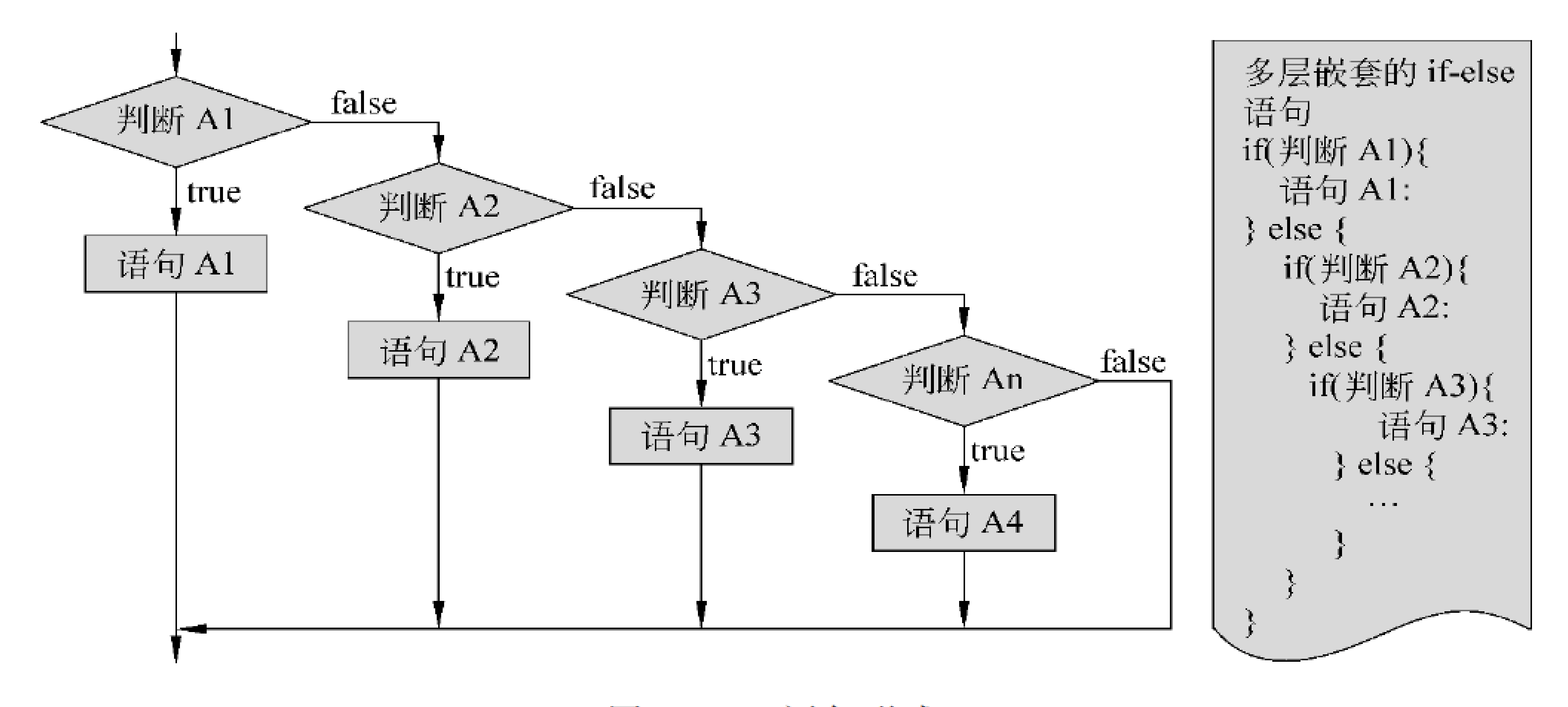


图 7-4 if 语句形式 2

2. switch 语句

对于图 7-4 所示的分支选择结构，无论有多少个分支，理论上说都可以使用嵌套的 if…else 语句表达。但是，if…else 语句的嵌套层次太多容易造成程序结构不清晰。Java 程序允许使用一种专门的多分支选择语句——switch 语句来表达这种多分支结构，这简化了多分支结构的表达，使程序简明易懂。这种语句的结构形式如下：

```
switch(整型变量){
  case  1:     一系列语句;break;
  case  2:     一系列语句; break;
  ⋮
  case  n:     一系列语句; break
  default:     一系列语句;
}
```

执行时系统计算出 switch 括号里的整形变量值，再根据这个变量值找到与它相匹配的 case 后面的值，执行那个值所在的 case 语句块，若无匹配则执行 default 语句块，其中 default 语句可有可无，若没有 default 语句则无匹配时什么都不执行。

3. for 语句形式

for 循环是编程中常常用到的循环结构，它结构清晰，流程简单，使用起来非常方便，for 语句的控制结构如图 7-5 所示。

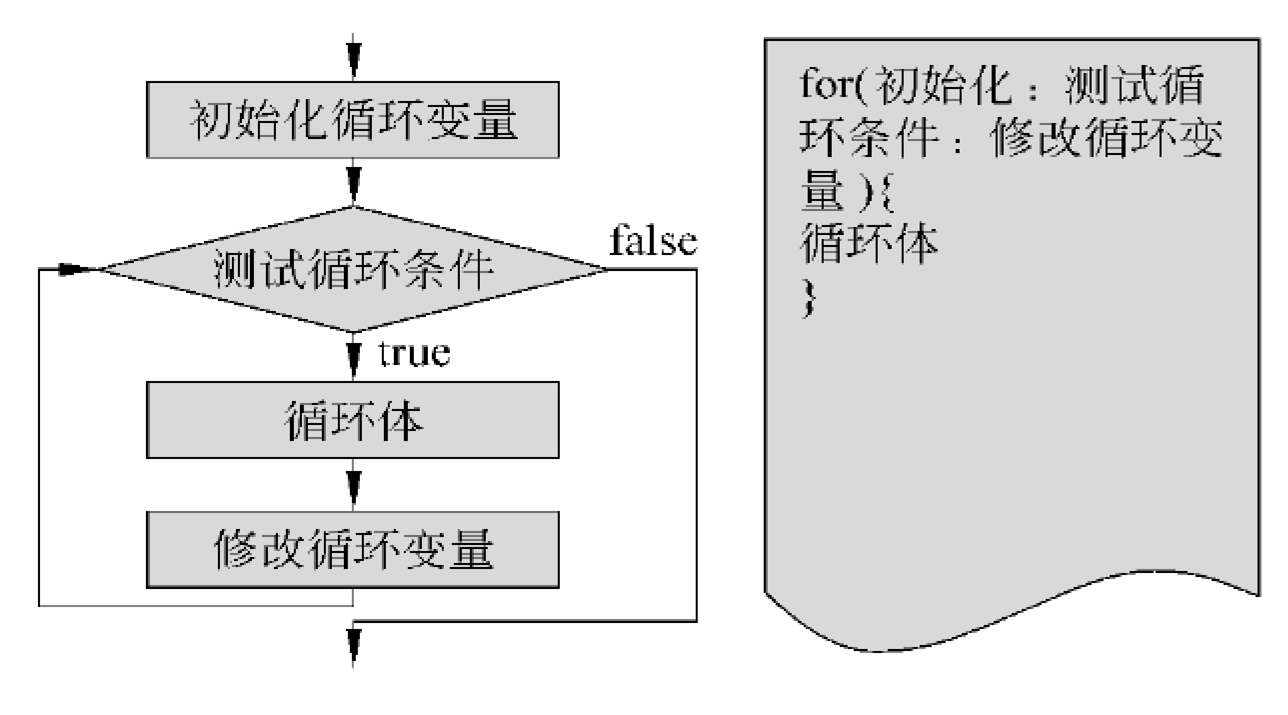


图 7-5　for 语句的控制结构

另外，Java 本身还增加了一个增强型的 for 循环结构，关于这个内容，由于篇幅原因，本书不对这个知识点再作深入的讨论，有兴趣的读者可以参考别的资料。

4. while 和 do…while 语句

Java 中循环还可以使用 while 和 do…while 语句，两种语句的控制结构如图 7-6 所示。while 和 do…while 语句在循环结构中所起的作用是一样的，但还有一些微妙的差别，while 语句是首先判断表达式再执行循环语句，do…while 语句是先执行循环语句再判断表达式，读者应当注意到这一点。

需要强调的是，在循环结构中，常会用到 break 和 continue 语句。Break 语句用于终止某个语句块(即 break 所在的{}内的代码段)的执行，用在循环语句体中，可以强行退出循环；而 continue 语句用在循环语句中时，它的作用是终止某次循环过程，跳过循环体中 continue 语句下面未执行的代码，开始下一次循环过程。

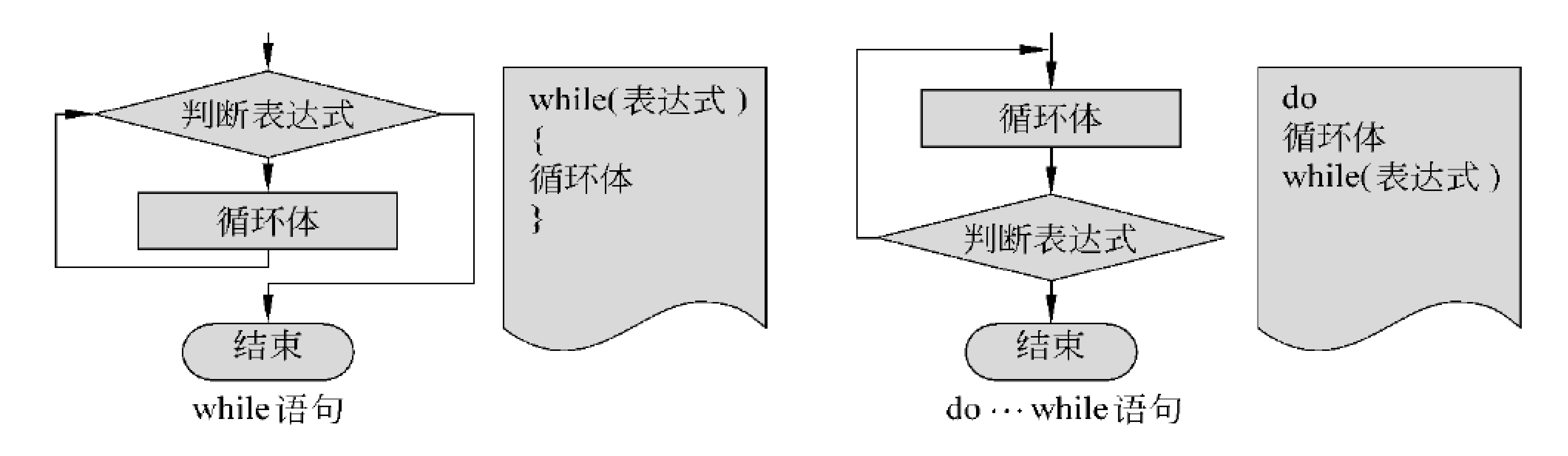


图 7-6　while 和 do…while 语句

7.3　Java 的面向对象编程基础

Java 是一种纯粹的面向对象程序设计语言，在 Java 看来，所解决问题中的所有的东西都是对象，程序是一大堆对象的组合，每个对象都有自己的存储空间每个对象都一定属于某个类，所以 Java 编程实际上就是对类和对象的编程。

7.3.1　类和对象

对象是系统用计算机语言对问题域中事物(一个实体)的描述，它是构成系统的一个基本单位。对象通过“属性(attribute)”和“方法(method)”来分别对应事物所具有的静态属性和动态属性。

类(class)是用于描述同一类型的对象的一个集合，是对同一类型的对象的一个抽象，类中定义了这些对象的静态属性和动态属性。

类与对象的关系就如模具和铸件的关系，类可以看成是一类对象的模板，程序根据这个模板“生成”不同的对象，各个对象可以看成该类的一个具体实例。

在日常生活中，常常会遇到类似类与对象这样的关系，例如一个公司的所有职员可以看作是一个类，即可以对这个群体作出抽象，抽象成类后然后对不同的职员可以生成不同的对象，当有新的职员进来时就可以根据这个类生成相应的唯一的对象，来标识新的职员，如果一个老职员要退出公司，这时把他所有对应的对象删掉即可。据此，我们可以画出公司职员的类和对象间的转换图如图 7-7 所示。这里“－”和“＋”表示该项是私有的或公有的，这是

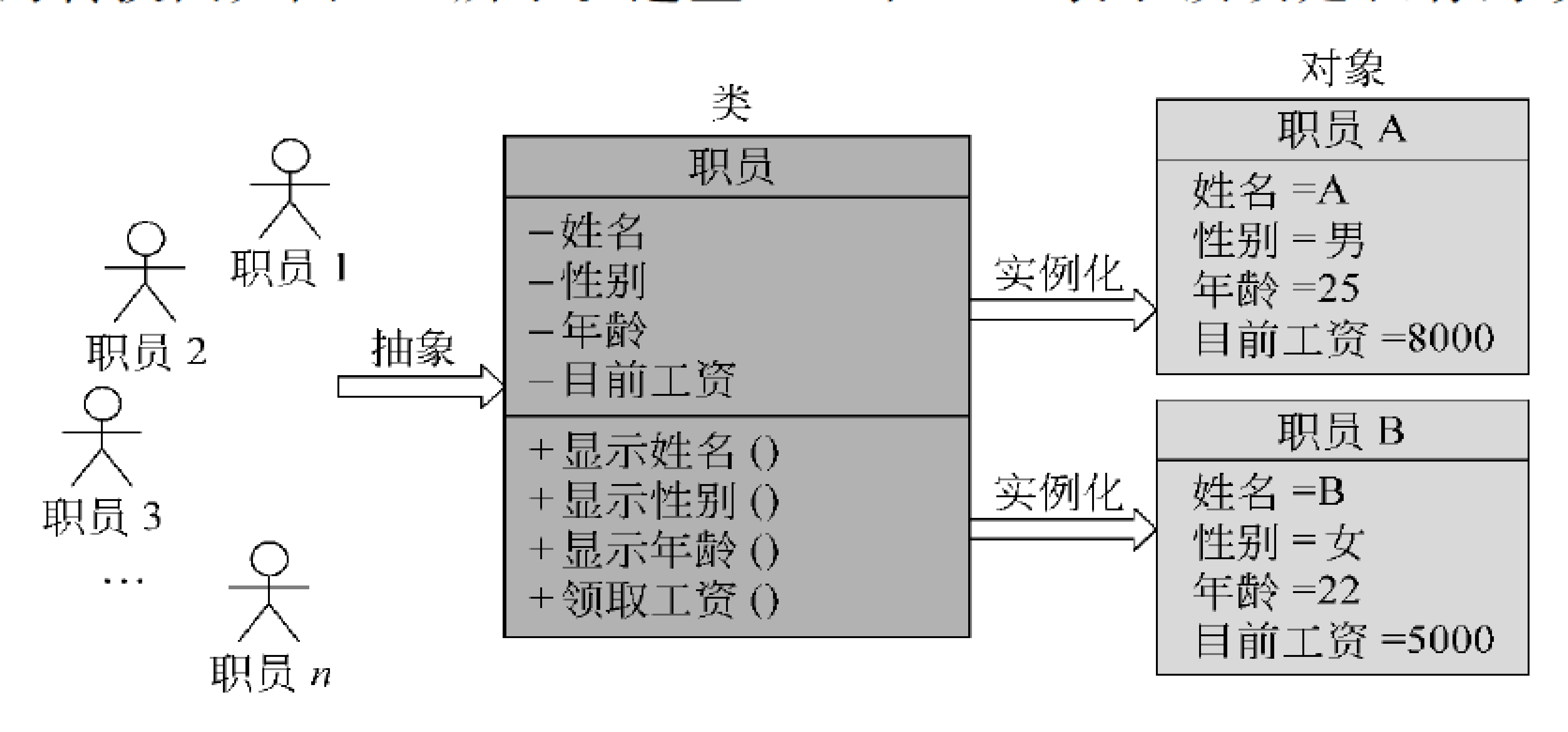


图 7-7　一个公司中职员的类和对象的转换

一种 UML(统一建模语言)类图中约定俗成的表示法。

在 Java 语言中,定义一个类的语法形式如下:

```
class 类名{
    成员属性;
    成员方法;
}
```

7.3.2 继承性

特殊类的对象拥有其一般类的全部属性与服务,称作特殊类对一般类的继承。简单地讲,继承就是:"a kind of"关系。例如,汽车是一种陆地交通工具,陆地交通工具是一种交通工具等。在 Java 语言中,通常称一般类为父类(superclass,超类),特殊类为子类(subclass)。例如在上面中,交通工具是父类,陆地交通工具是交通工具的子类;汽车又是陆地交通工具的子类。父类和子类的关系可以由图 7-8 可以看出,其中—▷表示继承的关系。

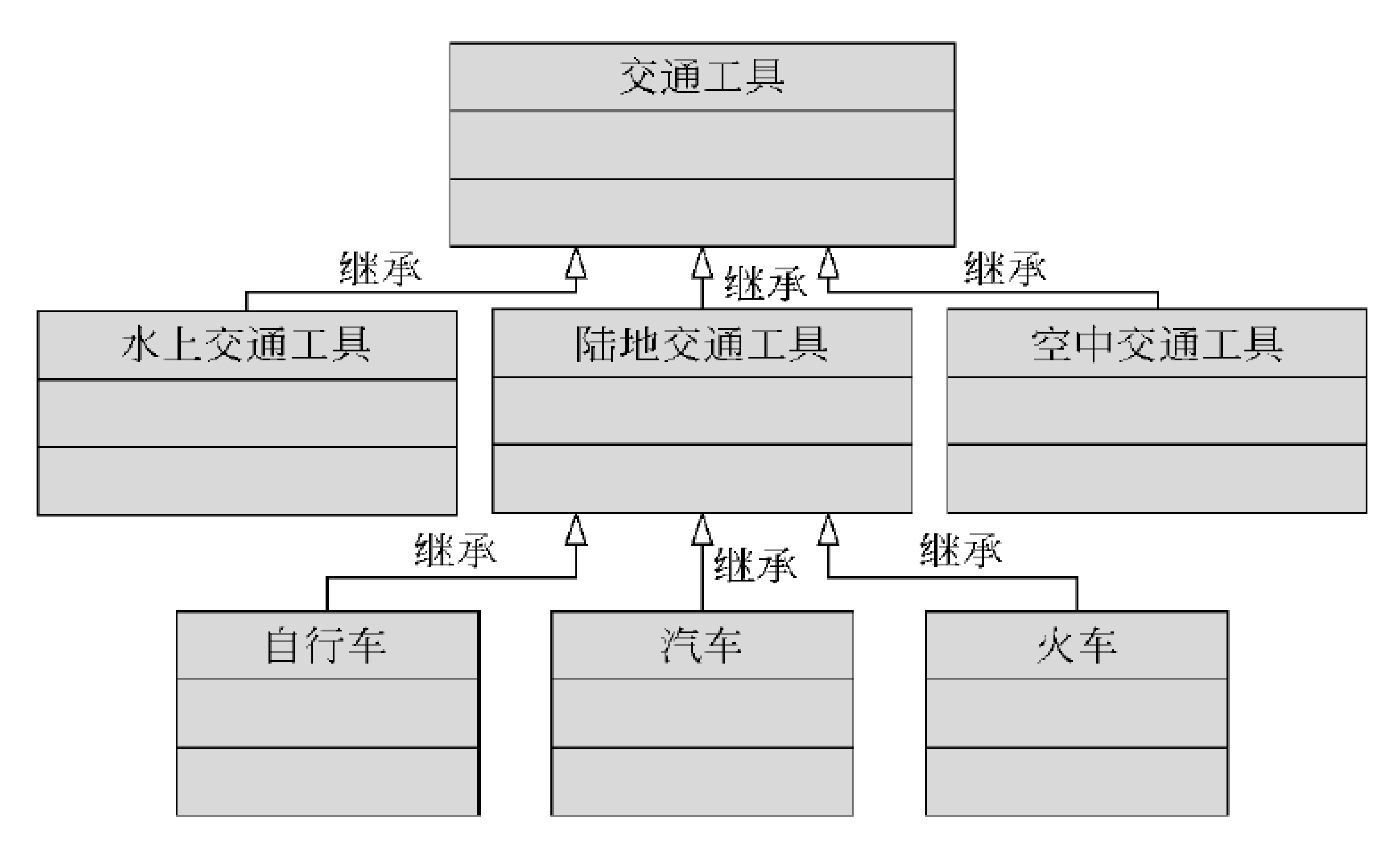


图 7-8　经抽象后得到的父类与子类间的继承关系图

继承是面向对象程序设计中一个十分重要的环节,只有对继承有了较好的理解,才能设计出良好的类组织。

在 Java 语言中,一个类 A 继承另一个已存在的类 B 的语法形式如下:

```
class A extends B {
    成员列表;
};
```

其中 B 为父类,A 为 B 的子类。花括号中的成员列表是派生类自己的成员。Java 继承机制的语法十分简单,只要在声明一个派生类的同时用保留字 extends 来指明它的父类就可以了。通过继承,子类拥有了父类的所有成员(成员变量和方法)。

注意:Java 只支持单继承,不允许多继承,这是 Java 与 C++ 不一样的地方。Java 中要实现多继承必须使用实现接口的方法。

7.3.3 包

Java 语言引入包(package)的机制使得 Java 更易于层次化地组织一个大型的 Java 程序。

程序包是多个类或接口的集合,这些类和接口可分别在不同的.java 文件中定义,每个文件称为一个编译单元。一个 Java 程序可以定义多个程序包,每个程序包又可以有多个编译单元或子程序包,每个编译单元又可定义多个类或接口,这是大型 Java 程序的典型组织结构。程序包、子程序包、编译单元、类或接口构成了 Java 程序的逻辑组织结构。

1. 定义包

定义包的语句必须是源文件的第一条语句,以指明该文件中定义的类所在的包。其格式如下:

```
package 程序包名;
```

如:

```
package ncu.jspbook.ch7;
```

Java 编译器把包对应于文件系统的目录管理,package 是定义包的保留字,在程序包中用“.”来指明包的层次,如上面的例句所表示的就是该文件的类位于.\ncu\jspbook\ch7 目录下。

2. 引用包

程序包使得 Java 程序的组织结构层次化,也使得类名层次化。当两个类处于不同的程序包时,一个类要访问另一个类就不能简单地使用类名来访问,而必须使用该类的全名,比如类 MyClass 在上面的 ncu.jspbook.ch7 包中定义了,则在另外的包中的类需要使用 MyClass 类时要写全包名和类名即 ncu.jspbook.ch7.MyClass,或在文件开头使用 import 语句如:

```
import ncu.jspbook.ch7.MyClass;        //只引入 MyClass 这个类
```

或

```
import ncu.jspbook.ch7.*;              //可引入包中的所有类和接口
```

只有这样,Java 编译器才能根据类名所提供的路径找到相关的类。

注意:可以不需要 import 语句就可直接使用 java.lang 中的类,因为这个 java.lang 包是 Java 编译器自动加上的。

3. 包级访问控制

程序包引入了包访问控制。具有包访问控制级别的成员只能由同一个程序包的类访问。包访问控制级别是 Java 语言的默认访问控制方式。当声明某个类成员时,如果不使用保留字 public、protected、private 中任何一个进行修饰,则该成员就具有包访问控制。表 7-4 列出了这 4 种修饰符的访问控制权限(其中 default 表示包访问控制),其中标有“Yes”的表明该成员能被相应范围下的其他成员访问。

程序包既减少了类或接口的名字冲突,又使得 Java 程序的重用更为方便,组织结构更为清晰。Java 编程是常用的包见表 7-5,读者可以直接打开 JavaAPI 文档查看更多的资料。

表 7-4　4 种修饰符的访问控制权限

修　饰　符	类内部	同一个包	子类	任何地方
private	Yes			
default	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

表 7-5　Java 中常用的包

包　名	功　　能
java. lang	包含一些 Java 语言的核心类，如 String、Math、Integer、System 和 Thread，提供常用功能
java. net	包含执行与网络相关的操作的类
java. io	包含能提供多种输入输出功能的类
java. util	包含一些实用工具类，如定义系统特性、使用与日期相关的函数
java. sql	包含实现数据库 JDBC 连接的类和接口

7. 3. 4　接口

前面说过，Java 是不支持多继承的，但是 Java 可以通过实现接口（Interface）来实现多继承。从本质上讲，接口是一种特殊的抽象类，这种抽象类中只包含常量和方法的定义，而没有变量和方法的实现。

1. 接口的声明

声明接口的语法与声明类的语法类似，但接口的数据成员只能是公有的静态常量数据，因此数据成员的声明不能使用除 public、static、final 以外的修饰符；接口的成员方法只能是公有的抽象方法，因此成员方法的声明不能使用除 public、abstract 外的修饰符。接口的数据成员默认认为是公有静态常量，成员方法默认认为是公有抽象方法，所以在声明接口的成员时也可以省略 public、static、final、abstract 之类的修饰符。程序清单 7-2 给出了接口 Pet（宠物）的声明：

程序清单 7-2：

```
//Pet.java
public interface Pet {
    public void setName(String name);    //宠物必须是可以命名的
    public String getName();             //宠物必须是可以叫出名字的
    public void eat();                   //宠物的进食方法
    public void sound();                 //叫声
    public void favor();                 //被宠爱的行为表现
}
```

2. 接口的实现

接口的实现需要用保留字 implements，多个无关的类可以实现同一个接口，而一个类

又可以实现多个接口，实现了接口的类必须重写接口中所有的方法。下面是实现接口 Pet 的例子，对应的实现关系如图 7-9 所示。

程序清单 7-3：

```
//TestPet.java
class Cat1 implements Pet {
    private String name;
    //必须重写接口的所有方法
    public void setName(String name) {
        this.name=name;
    }
    public String getName(){
        return this.name;
    }
    public void eat() {}
    public void favor() {
        System.out.println("眯眼...");
    }
    public void sound() {
        System.out.println("miao...");
    }
    public void catchMouse() {}
}
class Dog1 implements Pet {
    private String name;
    //必须重写接口的所有方法
    public void setName(String name) { this.name=name; }
    public String getName() { return this.name; }
    public void eat() {}
    public void favor() { System.out.println("摇尾巴..."); }
    public void sound() { System.out.println("miao..."); }
    public void swimming() {}
}
```

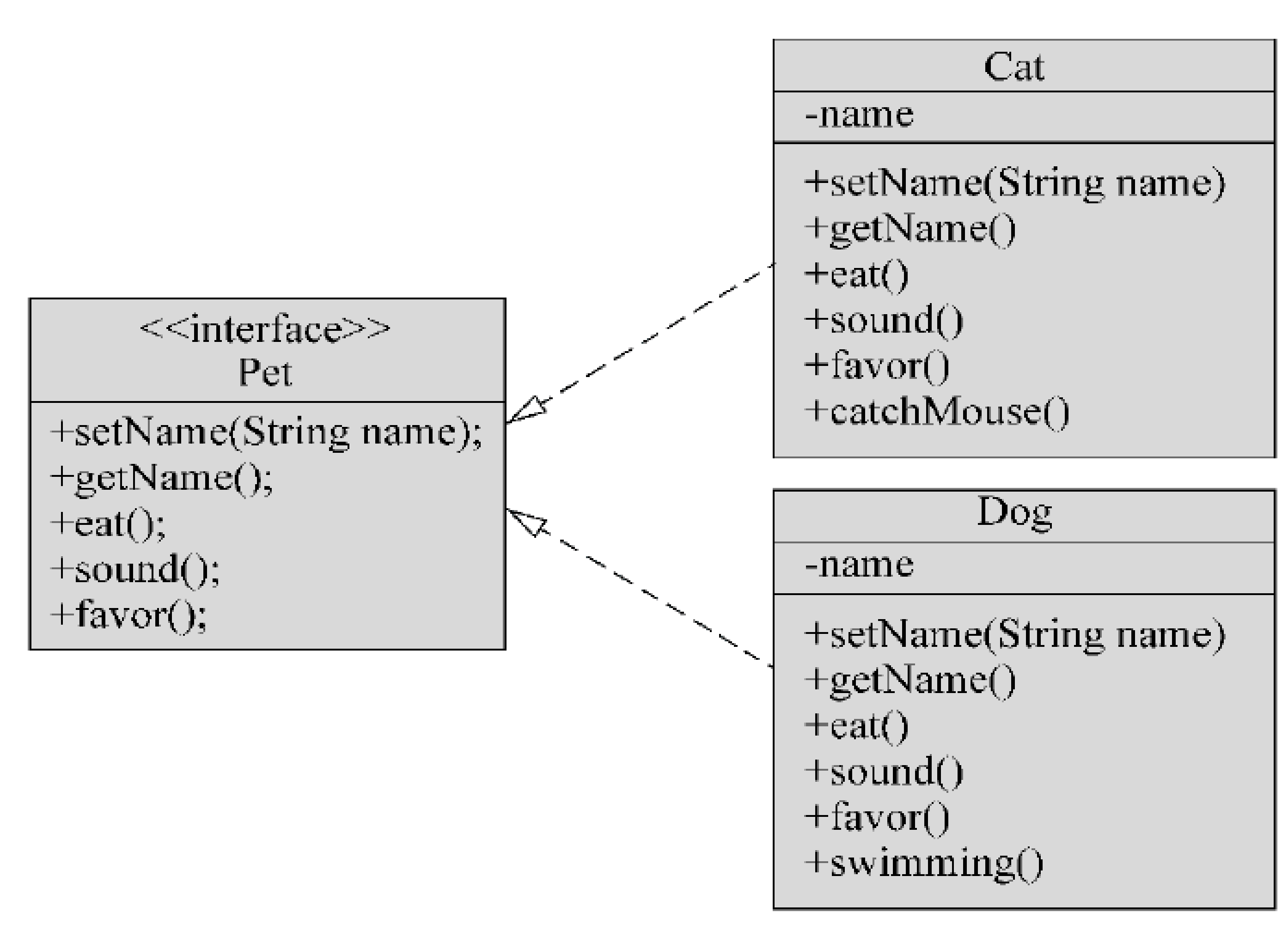


图 7-9　接口的实现

7.3.5 多态性

多态性就是程序中同一个符号在不同的情况下具有不同的意义。比如“/”这个符号，在操作数都是整数的时候表示整除，而如果操作数是浮点数的话就表示除法运算。多态性分为编译时多态性和运行时多态性，这里讲的主要是指运行时多态性，也叫动态绑定。事实上，多态性的原理是在程序运行期间判断所引用对象的实际类型，根据其实际类型调用其相应的方法。多态的存在必须具备 3 个条件。

（1）有继承：程序中存在子类对父类的继承（包括实现接口）。

（2）有重写：子类必须对父类中的必要方法进行了重写，以实现自己需要的功能。

（3）父类引用指向子类对象：程序调用的必须是父类，而实际上编译器却调用子类的对象。

显然，多态性带来很多好处，它可以提高程序的可扩展性，使得面向对象编程达到更高的层次，多态性是面向对象程序设计的核心。

程序清单 7-4 是利用继承抽象类来实现多态性的例子。

程序清单 7-4：

```
//Animal.java
public abstract class Animal {
    String color;
    String name;
    int state=1;                                    //默认设为活动状态
        Animal(String name, String color) {         //构造方法
        this.name=name;
        this.color=color;
    }
    void setState(int state) {                      //设定动物状态
        this.state=state;
    }
    int getState() { return state; }                //得到动物目前状态
    void run() { System.out.println("跑..."); }     //奔跑
}
```

程序清单 7-5：

```
// GirlsHavaPets.java
public class GirlsHavaPets {
public static void main(String[] args) {
        Cat cat1=new Cat("lingling", "白");
        Dog dog1=new Dog("jiajia", "黑");
        Dog dog2=new Dog("yuanyuan", "灰");
        Girl g1=new Girl("girl 1", cat1);
        Girl g2=new Girl("girl 2", dog1);
        Girl g3=new Girl("girl 3", dog2);
        dog2.setState(0);                           //设定 2 号狗的状态为睡着状态
```

```
            g1.petRun();
            g2.petRun();
            g3.petRun();
        }
}
class Girl {
    private String name;
    private Animal pet;                          //girl养了一个宠物
    Girl(String name, Animal pet) {
        this.name=name;
        this.pet=pet;
    }
    void petRun() {
        System.out.print(name+"'s pet: ");
        pet.run();                               //父类引用指向子类对象
    }
}
class Cat extends Animal {
    Cat(String name, String color) {
        super(name, color);
    }
    void run() {                                 //重写了父类的 run()方法
        if(this.state==1)
            System.out.println(this.name+"在跑!");
        else
            System.out.println(this.name+"已睡着!");
        }
}
class Dog extends Animal {
     Dog(String name, String color) {
      super(name, color);
     }
     void run() {                                //重写了父类的 run()方法
      if(this.state==1)
          System.out.println(this.name+"在跑!");
      else
          System.out.println(this.name+"已睡着!");
      }
}
```

运行结果如下：

```
girl 1's pet: lingling在跑!
girl 2's pet: jiajia在跑!
girl 3's pet: yuanyuan已睡着!
```

从运行结果可以看出，程序在运行时会自动地识别出当前的宠物是什么，需要怎样执行这个 run()方法，但是在 Girl 的对象看来，它只是使用了一个父类引用指向子类对象 pet.run()，或者说它只对 Cat 和 Dog 的父类 Animal 发出命令 petRun，至于到底是谁来执行就

交给编译器来完成了。

7.4 Java 的异常处理

7.4.1 异常与异常类

异常就是程序运行过程中遇到的严重错误，使程序运行中止，或者即使程序能够继续运行，但得出错误的结果甚至导致严重的后果。异常产生的原因很多，常见的有程序设计本身的错误、程序运行环境改变、软硬件设置错误等。Java 中常见的异常有“找不到特定的类”，“数组越界”，“除 0”等。

很多异常都是可以预见的，可以事先使用 if 语句来进行判断，以避免异常的发生，这是一种简单的解决办法。引入异常处理机制后，异常处理就可以使用统一的语句和接口，使异常处理更方便、更安全。

设计良好的程序应该在出现异常时提供处理这些错误的方法，使得程序不会因为异常的发生而阻断或产生不可预见的结果。

Java 异常类的结构分布如图 7-10。

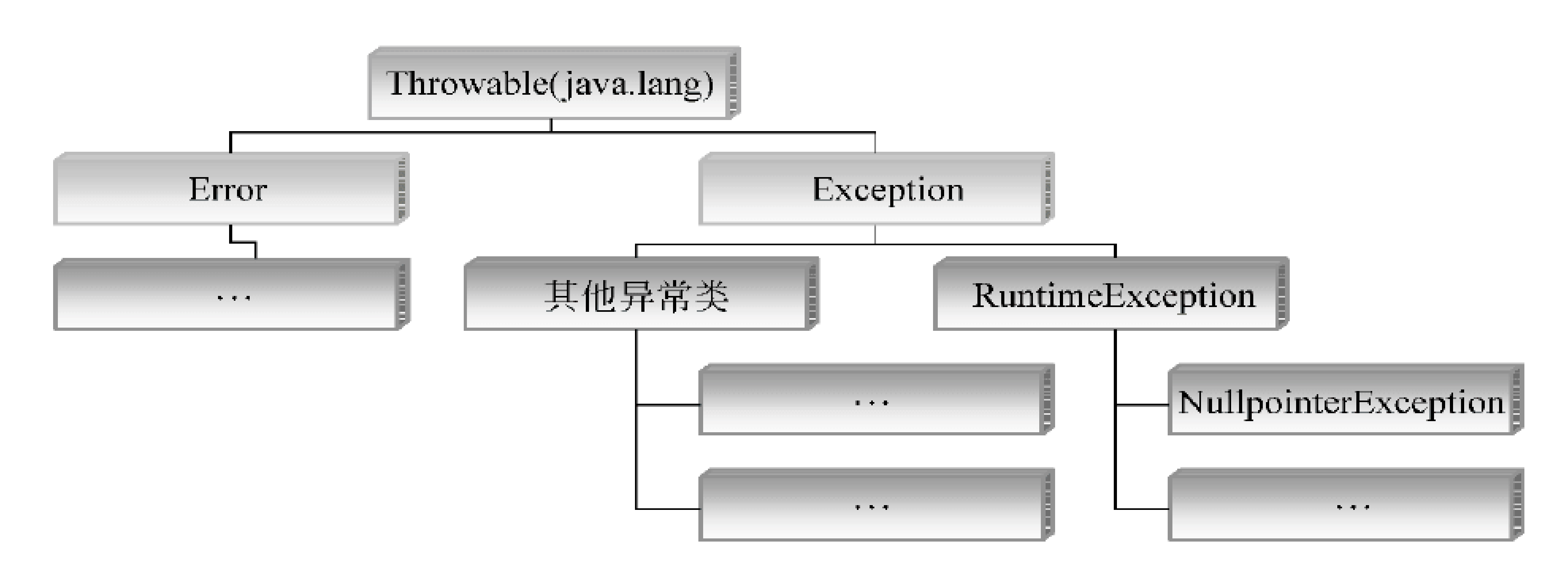


图 7-10 Java 异常类的结构图

（1）Error：称为错误，由 Java 虚拟机生成并抛出，包括动态链接失败、虚拟机错误等，程序无法处理这些错误。

（2）Exception：所有异常类的父类，其子类对应了各种各样可能出现的异常事件，一般需要用户显式的声明或捕获。

（3）RuntimeException：一类特殊的异常，如被 0 除、数组下标超出范围、操作空指针（对象为空）等，其产生比较频繁，处理麻烦，如果显式的声明或捕获将会对程序可读性和运行效率影响较大。因此由系统自动检测并将它们交给默认的异常处理程序而用户不必对其抛出和处理。

其他异常类中比较常见的有 IOException、InterruptedException、ClassNotFoundException、DataFormatException 和 SQLException 等，这些异常类常常出现在 Java 应用程序中，是一个 Java 程序员所应当熟知的。

7.4.2 异常的抛出

异常的处理分为异常抛出和异常捕获，本小节主要讲解异常的抛出。

Java 程序的执行过程如出现异常事件，可以生成一个异常类对象，该异常对象封装了异常事件的信息并将被提交给 Java 运行时系统，这个过程称为抛出(throw)异常。

程序中常用的异常抛出格式如下：

```
class A {      //throws 后面声明本方法中可能抛出的异常
    public void someMethod() throws Exception1[Exception2 ...] {
        ⋮
        //在程序中具体怎么抛出可以由 throw 和后面的一个异常类对象来确定
        if(…) throw new Exception1(传送特定字符串到异常类的构造方法);
        if(…) throw new Exception2(…);
        ⋮
    }
}
```

7.4.3 捕获异常

当 Java 运行时系统接收到异常类对象时，会寻找能处理这一异常的代码并把当前异常对象交给其处理，这一过程称为捕获(catch)异常。

异常捕获的结构如下(try catch finally)：

```
try {
    可能出现特定异常的代码;
} catch() {
    处理异常的语句 1;
} catch() {
    处理异常的语句 2;
} …
  finally {
    最终执行的语句;
}
```

在这个结构中，catch 子句可以有一个或者多个，但是如果有多个 catch 时其后面的异常类要遵循具体到一般的原则，即如果存在需要捕获父类和子类异常，必须首先捕获子类异常然后才能捕获父类异常而不能倒过来，否则编译时会出错；finally 子句可有可无，但前提是必须保证整个结构至少存在一个 catch 语句或者 finally 语句。

这个结构的执行有点像 switch case default 结构。如果 try 子句里某条语句抛出了异常，则停止执行后面的剩下的语句，直接跳到 catch 子句，检查抛出的异常和 catch 后面的异常类型是否匹配，不匹配则跳过当前的 catch 语句检查下一个 catch 语句中的异常类型，匹配则执行 catch 子句里面的语句。执行完毕以后，不会再回到出现异常的位置继续执行，而是会看有没有 finally 子句，有则执行 finally 子句然后跳出 try catch finally 结构，没有就直接跳出整个结构。如果 try 子句不发生任何异常，则程序会跳过所有的 catch 子句直接执行 finally 子句。

在 catch 中声明的异常对象(catch(SomeException e))封装了异常事件发生的信息，而在 catch 后的花括号中可以使用这个异常对象的一些方法获取这些信息，例如我们常用的

有以下两个方法。

（1）printStackTrace()方法：用来跟踪异常事件发生时执行堆栈里的内容；

（2）getMessage()方法：用来得到有关异常事件的信息。

另外，finally子句为异常处理提供了一个统一的出口，使得在控制流程转出到程序的其他部分以前，能够对程序的状态作统一的管理，通常是进行一些资源的清除工作，如关闭打开的文件、删除临时文件等，这样写可以达到控制流程清晰、程序执行安全的效果。

在编程时，可能会遇到这样一种情况，在声明方法时不愿或者无法对该方法所涉及的异常进行处理，由此Java语言采用了下面的异常处理机制。

在方法中如果对异常事件无法处理，则抛出(throws)这个异常然后留给它的上一级方法来处理，如果上一级再处理不了则再交给上一级，如此递推，到最后的最高一层方法时必须对这个异常作出捕获和处理。例如程序清单7-6。

程序清单7-6：

```
//TestExceptionSequence.java
import java.io.IOException;

public class TestExceptionSequence {
    public static void main(String[] args) {
        TestExceptionSequence test= new TestExceptionSequence();
        try {
            test.method1();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void method1() throws IOException {
        method2();
    }
    public void method2() throws IOException {
        method3();
    }
    public void method3() throws IOException {
        throw new IOException("IOException occur in method3");
    }
}
```

运行结果如下：

```
java.io.IOException:  IOException occur in method3
    at ch7.section7_4.TestExceptionSequence.method3(TestExceptionSequence.java: 21)
    at ch7.section7_4.TestExceptionSequence.method2(TestExceptionSequence.java: 18)
    at ch7.section7_4.TestExceptionSequence.method1(TestExceptionSequence.java: 15)
    at ch7.section7_4.TestExceptionSequence.main(TestExceptionSequence.java: 9)
```

本程序的运行原理如图 7-11 所示。

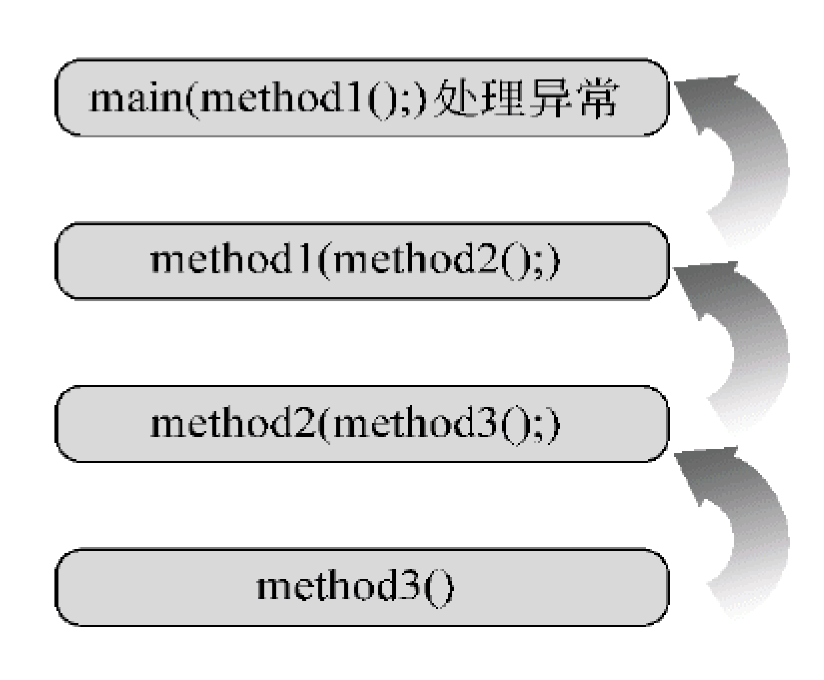


图 7-11 Java 异常处理实例

在设计程序时,如果有必要,用户也可以自定义相关的异常类,使用自定义异常类一般有如下步骤。

(1) 通过继承 java.lang.Exception 类声明自己的异常类。

(2) 在方法内部适当的位置生成自定义异常的实例,并用 throw 语句抛出。

(3) 在方法的声明部分用 throws 语句声明该方法可能抛出的异常。

Java 支持用户自定义的异常,这就给程序的安全性和可读性带来很大的好处,在程序设计时设计人员可以把主要精力放在程序的业务逻辑上。

7.5 Java 的多线程

C 语言中的程序执行都是一种顺序执行的方式,程序从 main( )函数开始,顺序的往下执行,遇到函数调用是就把执行权交给子函数直至子函数执行结束,然后子函数又把执行权交还给主函数,如此下去,直到主函数结束,整个过程就像一条线路,把各种操作串接起来,如图 7-12(a)所示。可以发现,程序在任何时刻只有一个执行点。

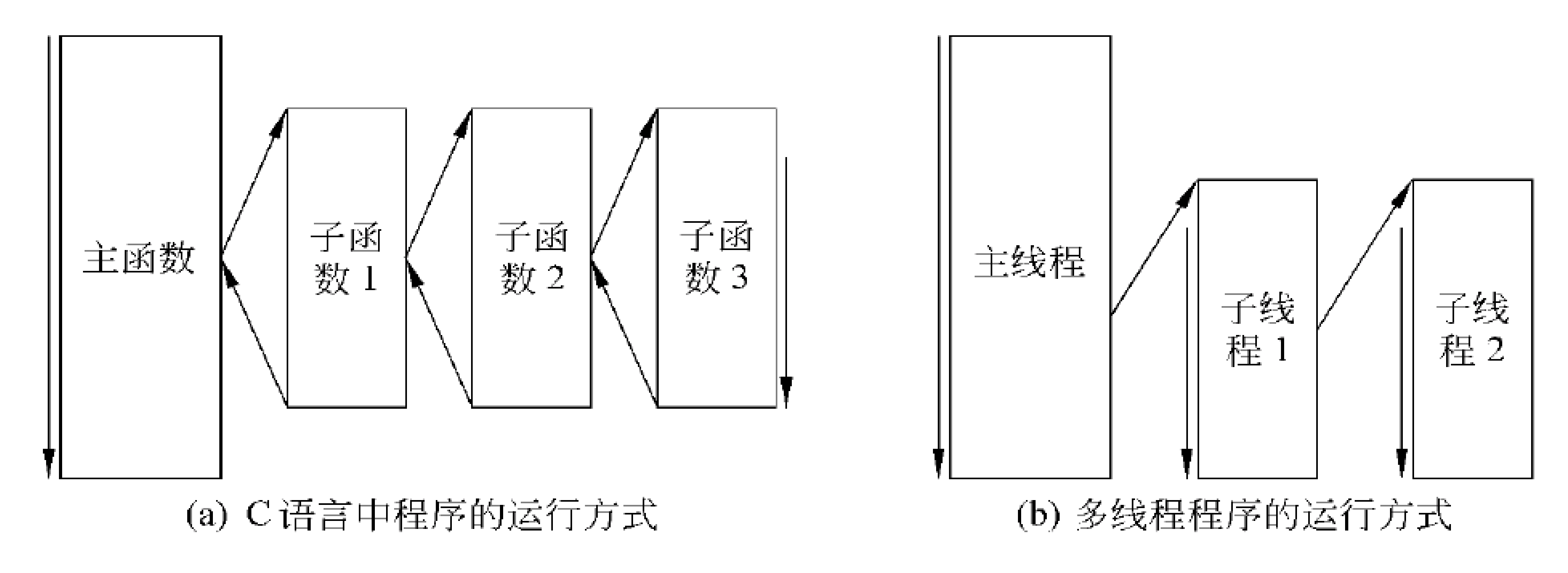


图 7-12 多线程与单线程的运行方式

单线程运行方式具有如下的特点。

顺序性:即程序执行过程可以看成是一系列严格按程序规定的状态转移的过程;

封闭性:也就是说程序执行的最终结果由给定的初始条件决定,不受外界因素的影响;

可再现性:只要输入的初始条件相同,则无论何时重复执行该程序都会得到相同结果。

与程序的顺序执行相对的是程序的并发执行,即一组逻辑上互相独立的程序或程序段在执行过程中,其执行时间在客观上互相重叠。程序的并发执行可以分成两种情况:一种是多道程序系统中多道程序的并发执行,此种情况下实际上是宏观上(程序级)同时进行,微观上(指令级)顺序执行的;另一种是在某道程序段的几个程序片段中,包含着一部分可以同时执行或顺序颠倒执行的代码。程序的并发执行是实现多线程技术的基础。

我们通常所说的进程是指一个具有独立功能的程序针对某个数据集的执行过程。线程与进程类似,是一段完成特定功能的代码。它是程序中单个顺序的控制流,也是一个进程内

的基本调度单位。线程和进程一样拥有独立的执行控制,并由操作系统负责调度。但同一进程可以包含多个线程,这些线程共享属于该进程的一块内存空间和一组系统资源;而线程自身的数据通常只有微处理器的寄存器数据,以及一个供程序执行时使用的堆栈。系统在产生一个线程,或者在各个线程之间切换时,负担要比进程小得多。此外,由于线程只是在单个进程的作用域内活动,所以线程间的通信也比进程简单。线程的实现要依靠操作系统,现代操作系统一般都支持线程技术。多线程的运行原理如图 7-12(b)所示。

线程与进程的区别如下:

(1) 每个进程都有独立的代码和数据空间(进程上下文),进程间的切换会有较大的开销。

(2) 线程可以看成是轻量级的进程,同一类线程共享代码和数据空间,每个线程有独立的运行栈和程序计数器(PC),线程切换的开销较小。

(3) 多进程是指在操作系统中能同时运行多个任务(程序),而多线程则是指在同一应用程序中有多个顺序流同时执行。

7.5.1 多线程的定义

多线程编程是指将程序任务分成几个并行的子任务,由这些子任务并发执行,一起协作完成程序的功能。多线程的执行是并发的,即在逻辑上是“同时”的,而不管是否是物理上的“同时”。如果系统只有一个 CPU,那么真正的“同时”是不可能的,而只能采用各线程轮流使用 CPU 的方法来模拟“同时执行”(只是由于 CPU 的速度非常快,用户感觉不到其中的区别);但是如果是在多 CPU 系统中,则多线程的并行执行是可能的,可以把不同的线程分配到不同 CPU 上同时执行。

Java 语言支持多线程,它的大部分类型都是在多线程下定义的,从而使整个系统成为异步系统。Java 通过 java.lang.Thread 类封装了线程及其上的操作。

1. 线程的创建和启动

线程的创建可以用两种方法:第一种是定义 Thread 的子类;另一种是实现 Runnable 接口。

(1) 继承 Thread 类并重定义 run()方法。定制一个线程在运行时执行的代码,可以通过定义一个 Thread 类的子类,并重定义从 Thread 中继承过来的 run()方法,使之执行指定的代码。run()方法是 Thread 中定义的一个空方法,定义格式如下:

```
class MyThread extends Thread {
    public void run() {… }                    //重写 run方法
}
```

当启动一个新线程时,run()方法中的第一条语句就是新线程将执行的第一条语句。线程启动的代码如下:

```
MyThread myThread= new MyThread(…);
Thread t= new Thread(myThread);
t.start();
```

(2) 定义线程类实现 Runnable 接口。Runnable 接口中只有一个 run 方法,任何类只要实现了 Runnable 接口系统就会自动识别出它为一个线程类。定义的格式如下:

```
class MyThread implements Runnable {
    public void run() {… }                              //重写 run方法
}
```

启动新线程的方法与上面相同。

一般的,如果要编写的类还要继承其他父类,此时就只能采用实现 Runnable 接口的手段,这也更符合面向对象的原则。

还有一点必须注意的是,无论采用上面哪种手段,都不能自己调用 run()方法,该方法只能是由主线程(或 JVM)调用。

2. 线程的生命周期

从创建一个新线程到这个线程消亡(或终止)的时间段称为线程的生命周期,在整个生命周期中,由于受到外界或线程自身需要的影响,线程表现出 5 种不同的状态。

(1) 创建状态。使用 new 运算符创建一个线程后,该线程仅仅是一个空对象,系统没有分配资源,称该线程处于创建状态(new thread)。

(2) 可运行状态。使用 start()方法启动一个线程后,系统为该线程分配了除 CPU 外的所需资源,使该线程处于可运行状态(Runnable)。

(3) 运行中状态。Java 运行系统通过调度选中一个 Runnable 的线程,使其占有 CPU 并转为运行中状态(Running)。此时,系统真正执行线程的 run()方法。

(4) 阻塞状态。一个正在运行的线程因某种原因不能继续运行时,进入阻塞状态(Blocked)。

(5) 死亡状态。线程结束后是死亡状态(Dead)。

各个状态间的互相转换共同组成了 Java 的线程运行机制,如图 7-13。表 7-6 列出了与线程相关的几个重要的方法,在访问和控制线程时常常会使用这些方法来完成。

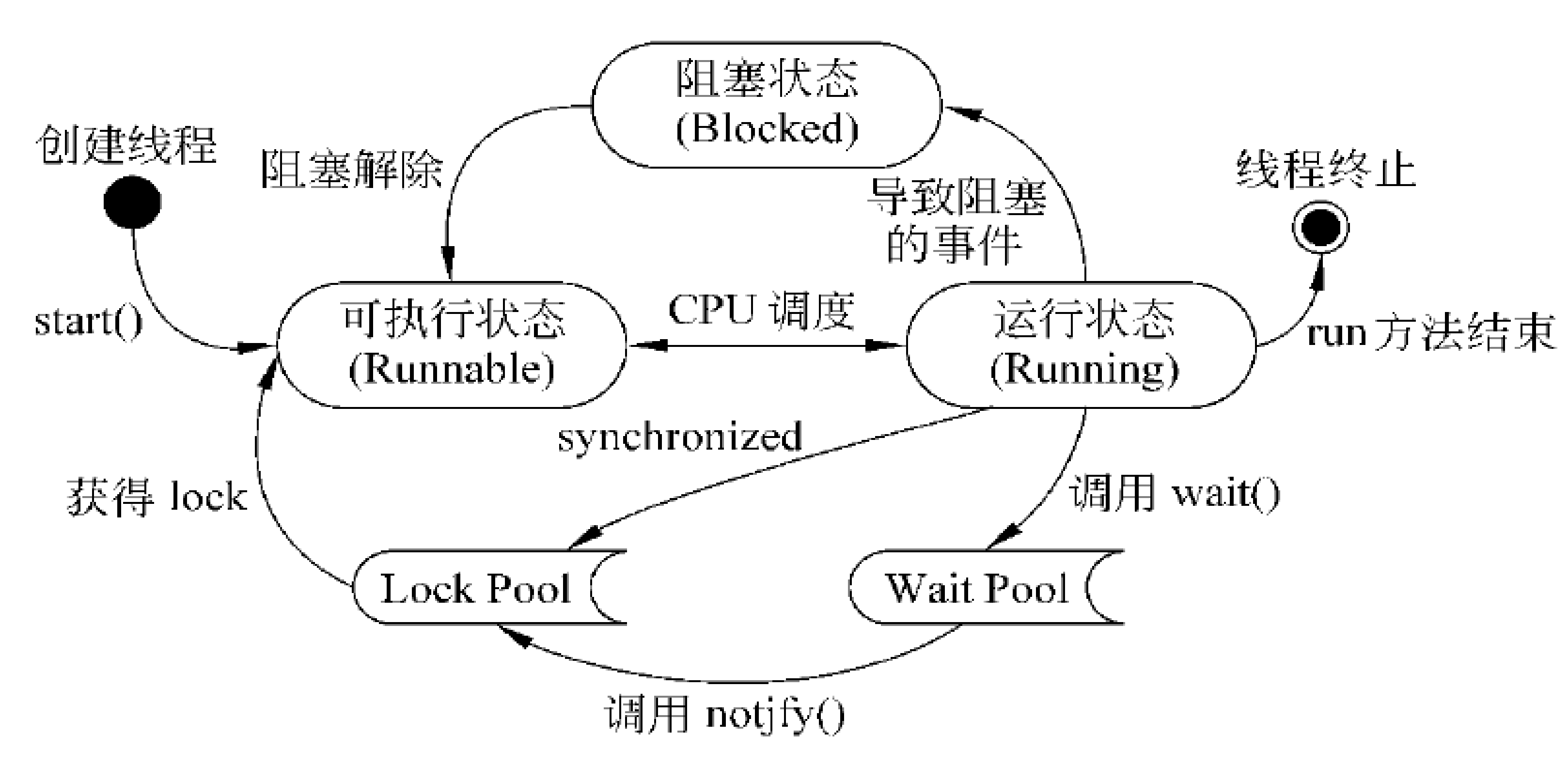


图 7-13 线程运行机制

表 7-6 与线程相关的方法

方　　法	功　　能
isAlive()	判断线程是否还活着,即线程是否还未终止
getPriority()	获得线程的优先级数值
setPriority()	设置线程的优先级数值

方　　法	功　　能
Thread. sleep()	将当前线程休眠指定毫秒
join()	调用某线程的该方法,将当前线程与该线程“合并”,即等待该线程结束在恢复当前线程的执行
yield()	让出 CPU,当前线程进入就绪队列等待调度
wait()	当前线程进入对象的 wait pool
notify()/notifyAll()	唤醒对象的 wait pool 中的一个或所有线程

注意:wait()方法与 sleep()方法虽然都有等待的功能,但两者是不同的,具体如下:

(1) sleep()方法是使线程停止一段时间的方法。在 sleep 时间间隔期满后,线程不一定立即恢复执行。这是因为在那个时刻,其他线程可能正在运行而且没有被调度为放弃执行,除非“醒来”的线程具有更高的优先级,或者正在运行的线程因为其他原因而阻塞。

wait()是线程交互时,如果线程对一个同步对象 x 发出一个 wait()调用,该线程会暂停执行,被调对象进入等待状态,直到被唤醒或等待时间到。

(2) 执行 wait()的线程不再拥有当前的锁,别的线程可以访问锁定的对象,而 sleep 是不可以的。

7.5.2 线程优先级

Java 提供一个线程调度器来监控程序中启动后进入就绪状态的所有线程,线程调度器按照线程的优先级决定应调度哪个线程来执行。

线程的优先级用数字表示,范围为 1~10,一个线程的默认优先级是 5。线程类中用三个常量记录了 3 种不同的优先级:Thread. MIN_PRIORITY、Thread. MAX_PRIORITY、Thread. NORM_PRIORITY,它们分别对应着优先级 1、10 和 5。

Java 中可以使用如表 7.6 的方法第二和第三个方法分别获得和设置线程对象的优先级。

7.5.3 线程同步

在编写多线程程序时,不可避免地会遇到多个线程并发访问一个对象或者数据的情况,此时就要十分小心地处理好线程之间的同步问题。处理不好线程同步将可能导致线程读到失效的数据,甚至引发死锁问题。

1. 没有同步可能带来的问题

为了演示并发访问同一对象时没有进行同步所带来的问题,请先看看下面的程序。

程序清单 7-7:

```
//SynDemo.java
public class SynDemo {
    public static void main(String[] args) {
        MyThread2 m1= new MyThread2();
        Thread t1= new Thread(m1,"t1");            //根据 m1 创建两个子线程
```

```
            Thread t2= new Thread(m1, "t2");
            t1.start();                                //启动两个线程
            t2.start();
        }
    }
    class MyThread2 implements Runnable {
        private int shareData= 0;                      //共享变量
        public void run() {
            Thread t= Thread.currentThread();
            for(int i= 0; i< 5; i++) {
                int copy= shareData;                   //将共享变量复制一份到 copy
                try {                                  //休眠若干时间
                    Thread.sleep((int)(Math.random() * 1000));
                } catch(InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Thread- "+ t.getName()+ ":  copy= "+ copy
                                  + "\tshareData= "+ shareData);
                shareData ++;
            }
        }
    }
```

该程序的功能很简单,主线程中创建了两个子线程,这两个子线程并发的访问同一个 Runnable 对象中的实例变量 shareData。在该 Runnable 对象的 run()方法中采用了一个迭代 5 次的 for 循环,在每次迭代时将 shareData 的值复制到临时变量 copy 中,之后该线程休眠了一段随机的时间,接着在输出 copy 和 shareData 的值。在只有一个线程的情况下,每次输出的 copy 值和 shareData 值应该是相同。但是,在多线程(两个子线程)环境下,该程序一种可能的输出(每次输出的结果是不同的)如下:

```
Thread- t1: copy= 0     shareData= 0
Thread- t2: copy= 0     shareData= 1
Thread- t2: copy= 2     shareData= 2
Thread- t1: copy= 1     shareData= 3
Thread- t2: copy= 3     shareData= 4
Thread- t1: copy= 4     shareData= 5
Thread- t2: copy= 5     shareData= 6
Thread- t1: copy= 6     shareData= 7
Thread- t2: copy= 7     shareData= 8
Thread- t1: copy= 8     shareData= 9
```

从上面结果可以看出,虽然每次循环时都将共享变量复制一份到 copy 中,但休眠一段时间后打印出来的 copy 的值已经改变了。正是由于这种多线程带来的不同步问题,使得在设计程序时不得不考虑如何来同步各个线程间的执行。

一般来说,出现不同步问题是由于对系统临界区的管理不当造成的。将程序中不允许

多个并发线程交叉执行的一段代码称为临界区。产生临界区的原因是属于不同并发线程的程序段共享公有数据。比如一个线程A进入临界区获取了有关数据，当这个线程A还没来得及对这些数据进行处理时，线程B又进入临界区，又使得临界区发生变化，而这时系统是没有任何察觉的，线程A在执行下一步时它还是使用目前的临界区数据，从而产生了不同步问题。

2. 同步问题的解决

Java语言中通常采用一种加锁机制来解决同步问题。在临界区前采用关键字“synchronized”来标识，它既可以是若干语句组成的语句块，也可以是一个方法。一旦一个对象中含有“synchronized”标识的语句块或者方法时，Java运行环境将为该对象分配一个唯一的对象锁。无论是哪个线程要进入对象的临界区，都必须先竞争得到该对象的对象锁；如果线程竞争不到对象锁，则线程将会进入阻塞状态，直到其他线程释放了对象锁从而重新唤醒它，让它再一次参加对象锁的竞争。

加锁可以有3种方式。

(1) 对临界区所在的方法加锁。例如上面的例子可以将MyThread2中的run方法加锁，把代码改成：

```
public synchronized void run() {… }
```

这样相当于线程t1进入run方法后把run这个代码段锁住，直至它运行完毕再把锁交出来，这时线程t2才能拿到这个锁进入run方法执行。

其运行结果如下：

```
Thread-t1: copy=2     shareData=2
Thread-t1: copy=3     shareData=3
Thread-t1: copy=4     shareData=4
Thread-t2: copy=5     shareData=5
Thread-t2: copy=6     shareData=6
Thread-t2: copy=7     shareData=7
Thread-t2: copy=8     shareData=8
Thread-t2: copy=9     shareData=9
```

(2) 对对象加锁。有时当互斥访问某个对象的语句相对于整个方法来说较少时，可能不希望把整个方法定义为同步方法，而是让里面的若干语句构成一个同步块，这时可以用“synchronized”来为该对象加锁。语法形式如下：

```
synchronized(obj) {
    //若干语句序列;
}
```

(3) 对类加锁。有时希望某些类中的一些类方法(静态方法)必须由各线程互斥执行，则可以把这些方法声明为同步的。这样做实际上是为相应的类加了锁，使得该类中的类方法在任意时刻最多只能由一个线程执行。为类加锁的一个例子是定义单实例类，这样的类只能被实例化一次。

注意：

(1) 死锁问题。Java 语言提供的同步机制虽然方便灵活并且功能强大，但它仍不能解决多线程编程时必须谨慎面对的所有问题，例如死锁问题。所谓死锁，是指各并发线程彼此互相等待对方所拥有的资源，但这些并发线程在得到对方的资源之前不会释放自己所拥有的资源，从而造成大家都想得到资源而又都得不到资源，各并发线程不能继续向前推进的状态。在多线程编程中，同步问题处理不当将很容易造成死锁问题。

(2) 何时需要同步。实现线程的同步需要很多额外的系统资源，此外由于要对对象进行锁操作，所以过多的使用同步机制会降低程序的效率。所以，是否采用同步机制要综合衡量各方面的需要。一般的，当有多个线程要修改同一个数据时，则可以把修改公用数据的方法声明为同步，只读取公用数据的方法可以不是同步方法。

小　　结

本章主要针对 Java 语言的基础知识做了简要的介绍，使读者在学习 JSP 以前能对 Java 语言有一个大概的了解。在这一章中介绍了 Java 的基本语法结构，面向对象的程序设计方法，异常的处理和 Java 中多线程的设计。完全靠这一章的内容学会和掌握 Java 编程是不可能的，如果读者需要系统的学习 Java，应该去参考相关的 Java 书籍。

练　习　7

1. 填空题

(1) Java 语言具有哪 5 个重要特点：________、________、________、________和________。

(2) Java 中的简单数据类型包括________、________、________和________，复合数据类型包括________、________和________。

(3) Java 的 3 个重要特性分别是________、________和________。

(4) 所有 Java 类的父类是________，所有 Java 异常类的父类是________，所有 Java 线程类的父类是________。

(5) Java 使用保留字________来标识该类继承了某个类，使用________来标识该类实现了某个接口，使用________来标识该程序块是互斥访问的。

(6) 实现多态的三个前提条件是________、________和________。

2. 选择题

(1) 下面程序段的执行结果是什么？(　　)

```
abstract class MineBase {
    abstract void amethod();
    static int i;
}
public class Mine extends MineBase {
    public static void main(String argv[]){
```

```
        int[] ar=new int[5];
        for(i=0;i<ar.length;i++)
        System.out.println(ar[i]);
    }
}
```

A. 打印 5 个 0。

B. 编译出错,数组 ar[]必须初始化。

C. 编译出错,Mine 应声明为 abstract。

D. 出现 IndexOutOfBoundes 的异常。

(2) 下面程序段执行的结果是什么?(　　)

```
public class Foo {
  public static void main(String[] args) {
    try{      return;
          } finally {
           System.out.println("Finally");
    }
  }
}
```

A. 程序正常运行,但不输出任何结果。

B. 程序正常运行,并输出 "Finally"。

C. 编译能通过,但运行时会出现一个异常。

D. 因为没有 catch 语句块,所以不能通过编译。

3. 实验题

编写一个 Java 应用程序,要求在程序中实现多态。

第 8 章　JSP 简介

8.1　了　解　JSP

JSP(Java Server Pages)是由 Sun 公司发布的一种 Web 开发技术，随着网络服务的不断发展，现在已经成为主流的 Web 程序开发技术之一，它具备了跨平台、通用性好、安全可靠等特点。

JSP 是由 Servlet 技术发展而来的，相比 Servlet，JSP 程序书写表示页面更为简单，同时它又具有 Servlet 的强大功能，因此它已经更加广泛地用于 Web 页面程序的开发。同时 JSP 也是基于 Java 的，因此具有良好的伸缩性，在网络数据库开发中拥有很大的优势。

8.1.1　JSP 的工作原理

JSP 是一种服务器端的 Web 程序开发技术。JSP 页面程序由 HTML 或 XML 标记和 JSP 脚本共同组成，文件以.jsp 作为扩展名存放在服务器上。那么 JSP 程序是如何被执行的？下面我们介绍 JSP 的工作原理。

当用户从客户端向服务器发出请求要首次访问某个 JSP 文件后，服务器会在磁盘上查找到该 JSP 文件，通过 JSP 引擎解释该文件代码，生成一个同名的 Java 文件，这个文件就是 Servlet；然后将该文件编译生成 Java Class 字节码文件，它会驻留在服务器中，当下次再要访问同一个 JSP 文件时，Servlet 引擎会直接调用该 class 文件执行而不需要重新编译；当 Servlet 引擎执行完该 class 文件后，服务器会将执行后生成的 HTML 文件返回给客户端由浏览器显示给用户。因此我们在客户端是看不到页面中的 JSP 代码的。

8.1.2　JSP 的特点

随着 JSP 技术的不断发展，它已经越来越多地应用在 Web 程序开发中。而且 JSP 可以和 EJB、J2EE 等组件进行集成，开发结构和功能更为复杂的企业级应用程序。加上它具有跨平台、执行速度较快的优势，因此在很多方面体现出其优越性。我们可以总结 JSP 的特点如下。

(1) 将内容的生成和显示分离。使用 JSP 开发页面，设计人员可以使用 HTML 或 XML 标记来设计和规划最终显示的页面，而使用 JSP 或脚本代码来生成页面中的动态显示内容，动态内容通常被封装在 JavaBean 中。这样可以使开发人员的分工更加明确，Web 程序更易维护，页面设计人员即使修改页面文件也不会影响内容的生成。

(2) 生成可重用组件。很多 JSP 页面程序都依赖可重用组件如 EJB、JavaBean 等来执行更为复杂的操作，程序开发人员可以共享这些通用的组件，这样可以加速整个程序的开发过程。

(3) 采用标识简化页面开发。JSP 标记具有扩充性，它允许用户定制常用的功能标记

库。由于 Web 页面设计制作者可以使用标记库中的标记，因此很大程度上减少了页面制作的复杂度。

（4）安全性。所有 JSP 代码都在服务器端运行，通过 JSP 引擎编译成 Class 文件后再执行，生成的动态内容会以 HTML 或 XML 的形式发回客户端浏览器，在客户端是看不到 JSP 代码的。另外由于 JSP 是基于 Java 的，具有很多的 Java 语言的特性，当然也包含 Java 的安全性，因此 JSP 也具有很好的安全性。

（5）跨平台。随着 Java 语言的迅速发展，几乎所有的平台都支持 Java，这也给 JSP 带来了广泛的适应性。

8.2 Tomcat 服务器的安装和配置

想要在服务器端执行 JSP 文件，就需要安装相应的运行环境。目前较为流行的 JSP 运行环境有 Tomcat、BEA Weblogic 和 IBM WebSphere 等。由于本书选择 Tomcat 作为 JSP 程序的运行环境，下面就来介绍 Tomcat 的安装、配置方法。

8.2.1 Tomcat 服务器的安装

在安装 Tomcat 服务器环境前，应先在系统中安装 Java 运行环境 JDK。本书以 JDK 6 作为 Java 运行环境，读者可以访问网址 http://java.sun.com 获取 JDK 6 的安装程序并进行免费下载，下载好安装程序后可以在本机上安装和配置（JDK 环境的安装和配置在此不详述）。

配置好 JDK 环境后，接下来就是 Tomcat 的安装。本书以 Tomcat 6.0 作为 JSP 运行服务器，读者可以访问网址 http://tomcat.apache.org 获取 Tomcat 6.0 的安装程序并免费下载和使用。由于 Tomcat 的官方网站提供了几种不同的安装程序，本书选择的是基于 Windows 的安装程序，它是一个可执行文件，我们只要运行该应用程序就可以按照安装向导进行服务器环境的配置。

安装和设置 Tomcat 的基本步骤如下。

（1）运行安装程序，会看到如图 8-1 所示界面。



图 8-1 运行安装程序首界面

(2) 单击 Next 按钮，将进入授权界面，选择 I Agree 按钮，则进入组件选择界面。在该界面中可以选择要安装的组建，通常选择全部安装，如图 8-2 所示。然后单击 Next 按钮。

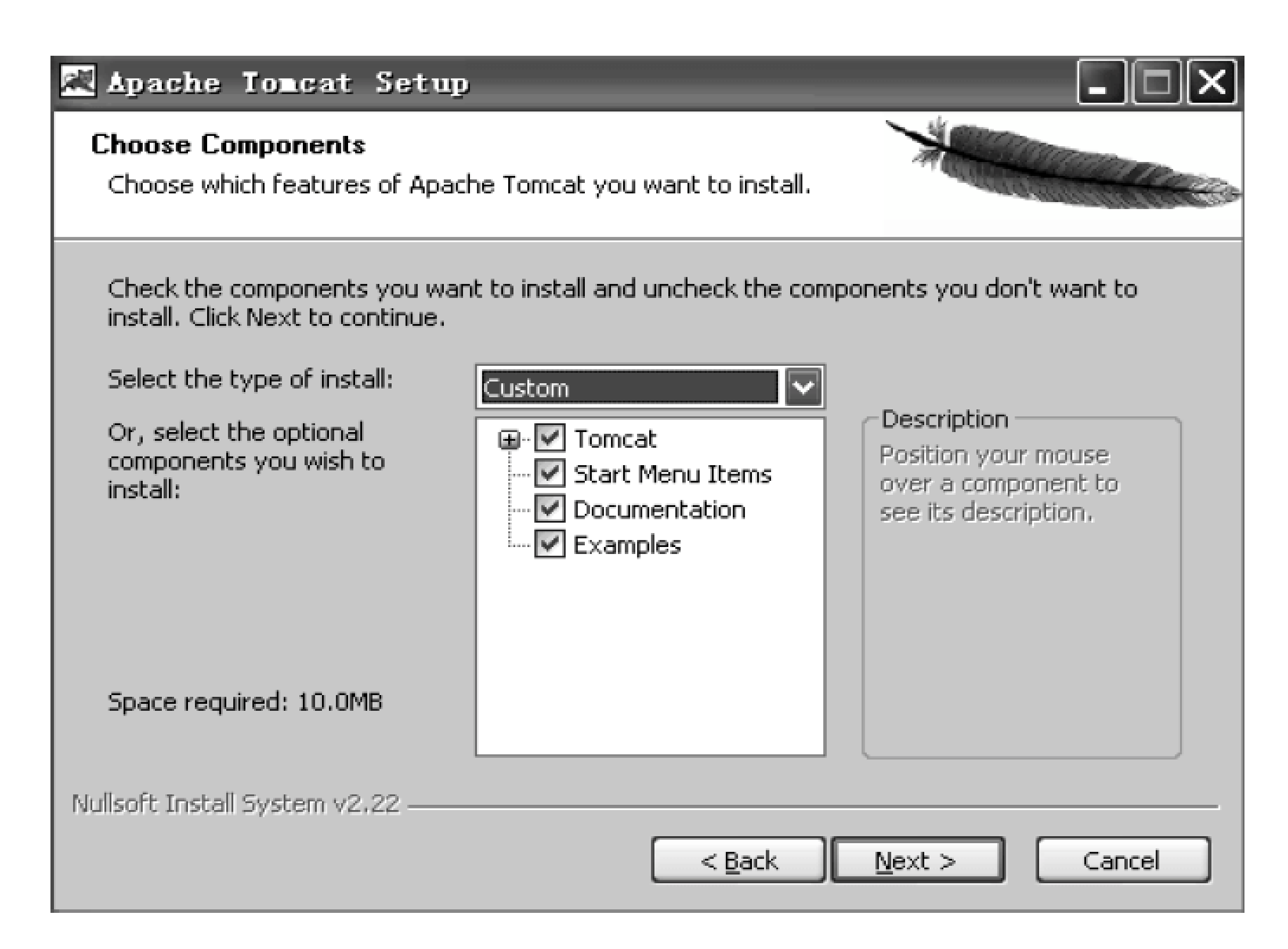


图 8-2　组件选择

(3) 选择要安装的 Tomcat 文件目录，如图 8-3 所示，然后单击 Next 按钮。

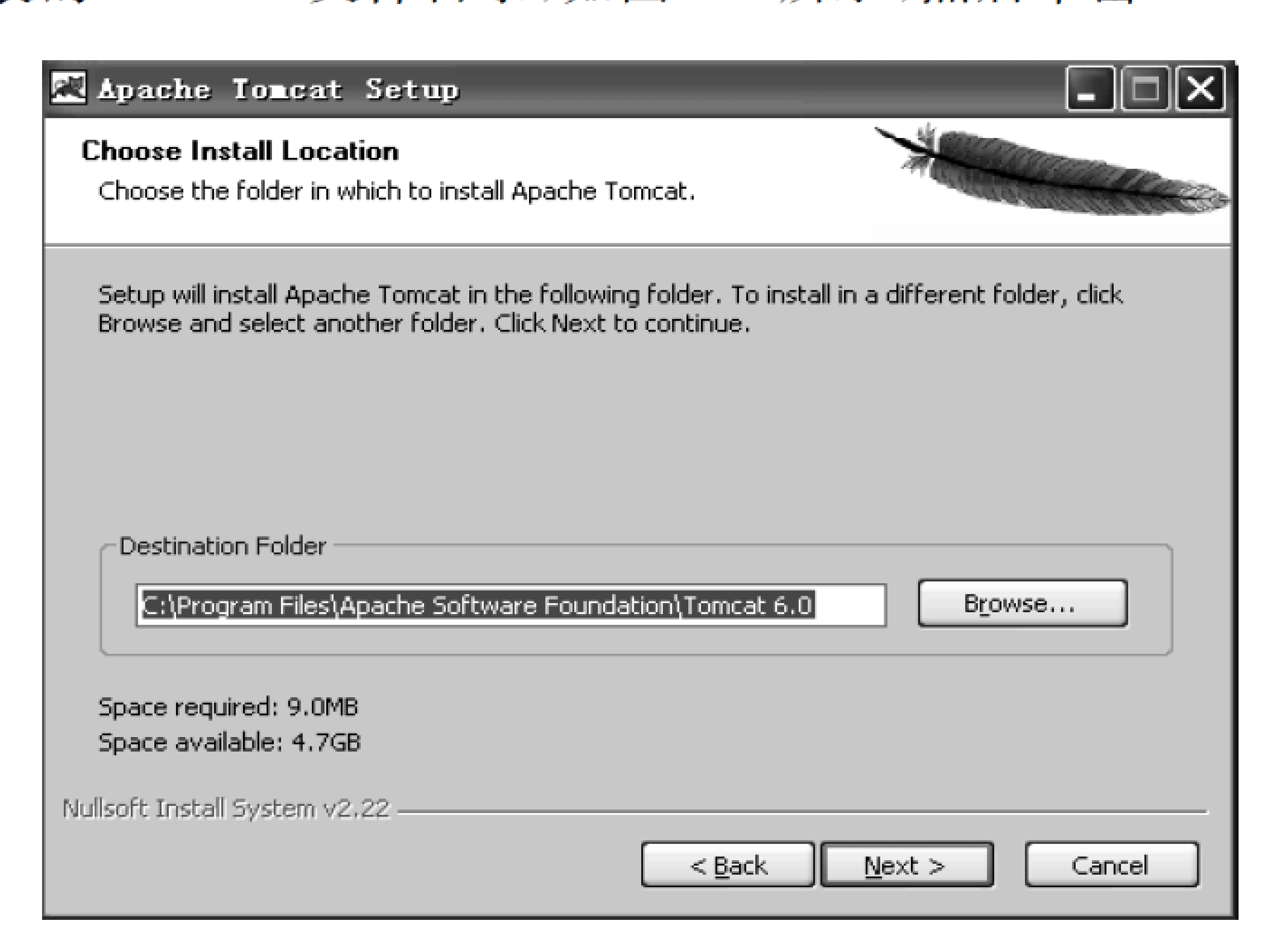


图 8-3　设置安装路径

(4) 设置 HTTP 访问端口，默认为 8080，读者也可以根据需要修改为其他端口号，但所使用的端口号不能被其他应用程序占用。设置管理员用户名和密码，如图 8-4 所示，然后单击 Next 按钮。

(5) 设置 JDK 安装路径，如图 8-5 所示。如果 JDK 已经安装好，Tomcat 会自动找到 JDK 的 JRE 所在目录，单击 Install 按钮。

(6) 安装完成后单击 Finish 按钮结束安装。

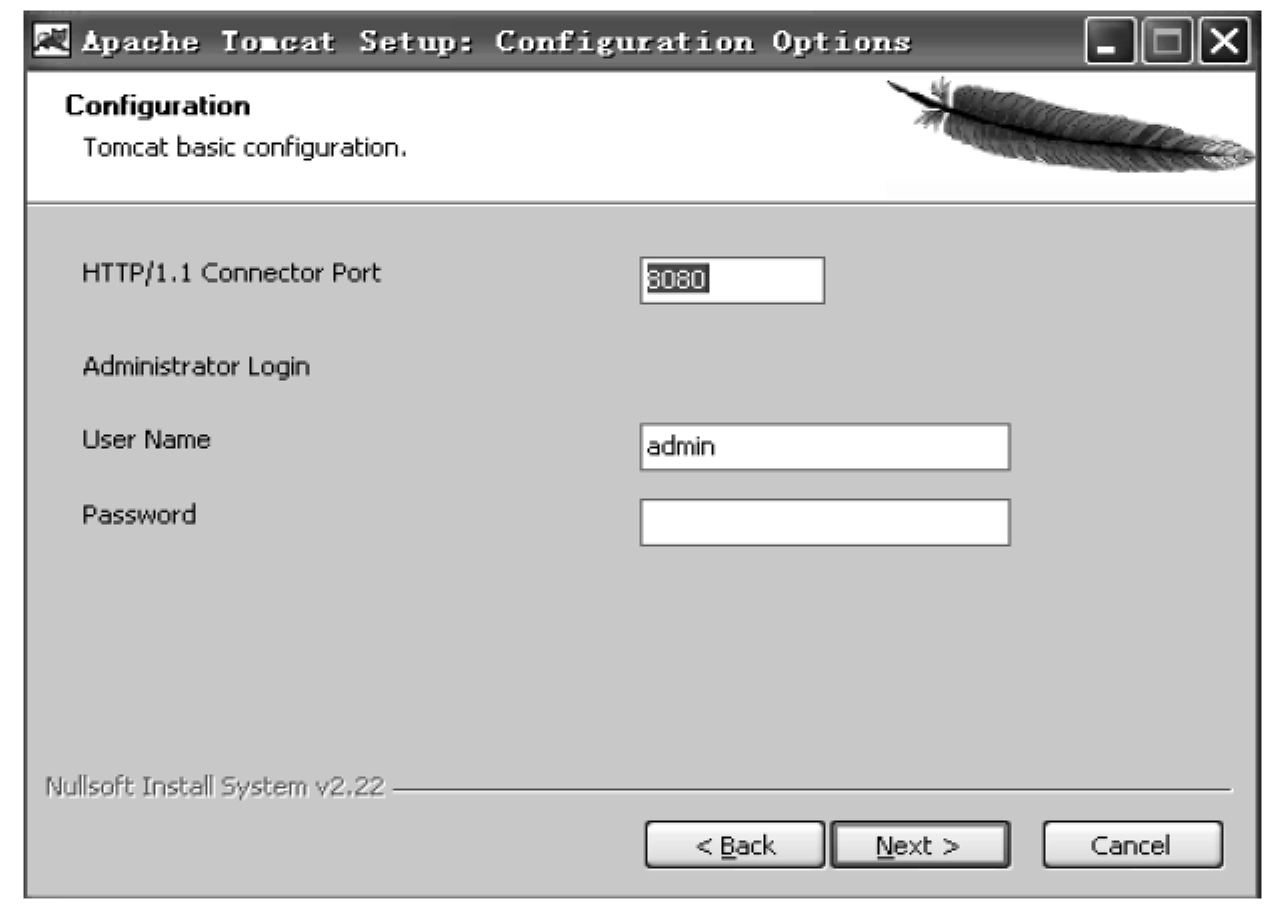


图 8-4　设置端口、管理员用户名和密码

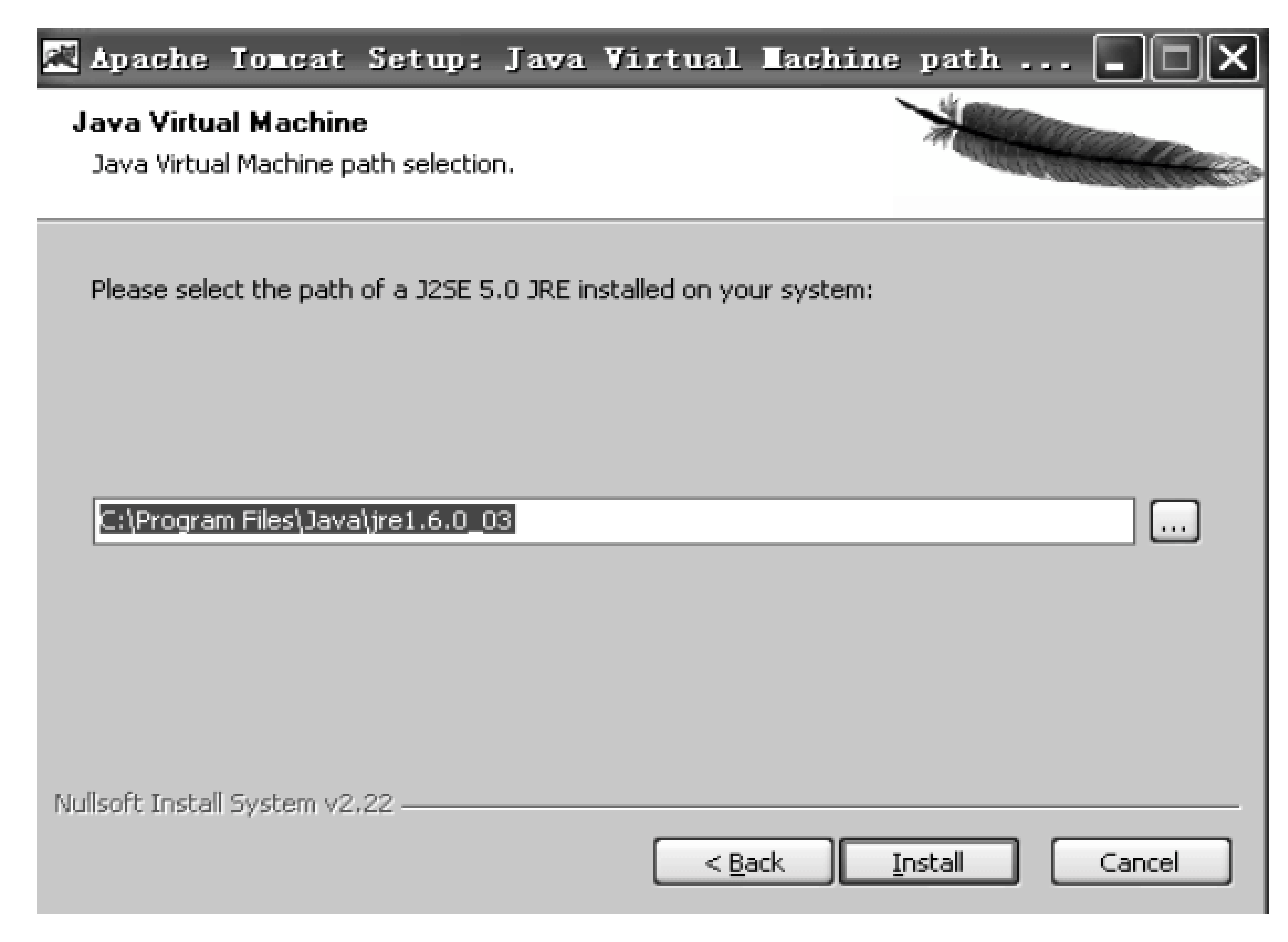


图 8-5　设置 JRE 所在目录路径

8.2.2　Tomcat 服务器的配置和测试

安装好 Tomcat 后，可以选择"开始"|"程序"|"Apache Tomcat 6.0"|"Configue Tomcat"，在弹出的对话框中单击 Start 按钮启动 Tomcat，如图 8-6 所示。可以打开浏览

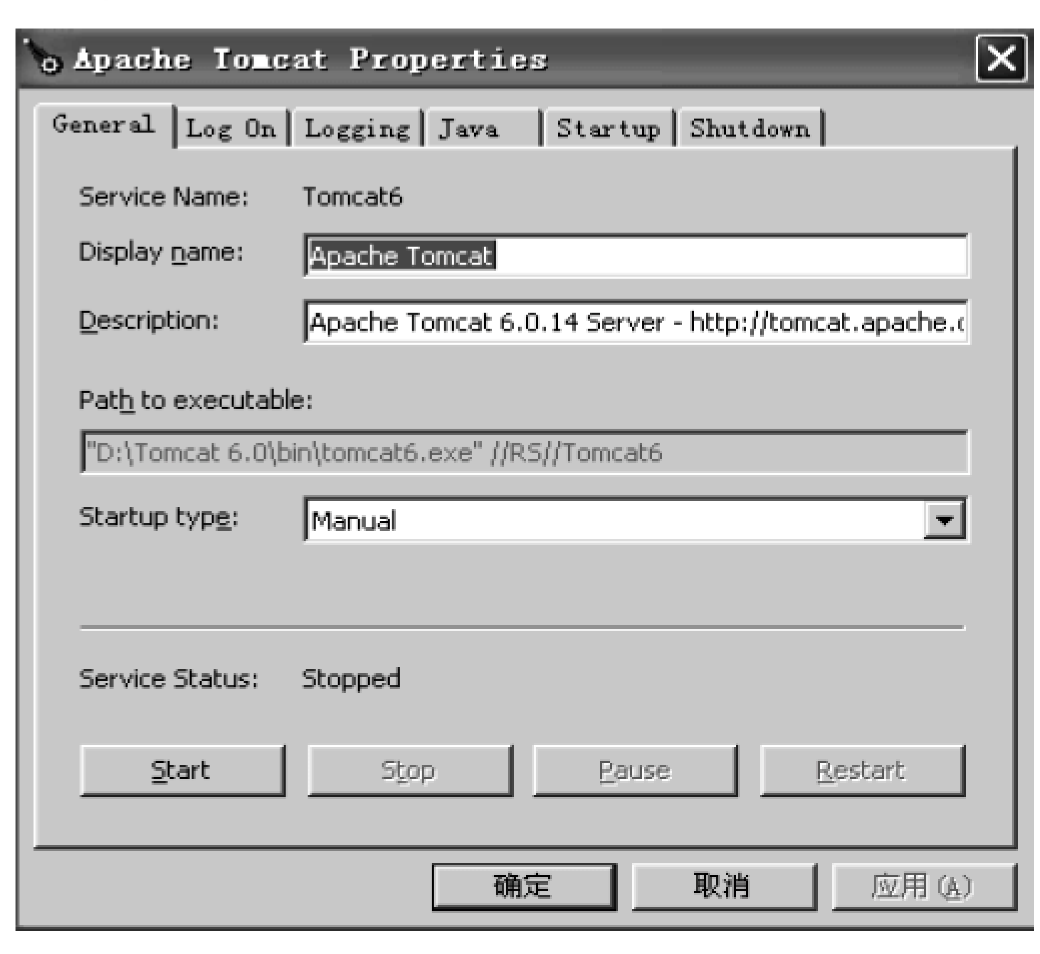


图 8-6　Tomcat 默认主页面

器，在地址栏中输入 http://localhost:8080，如果 Tomcat 安装成功了，浏览器会显示如图 8-7 所示的运行结果。

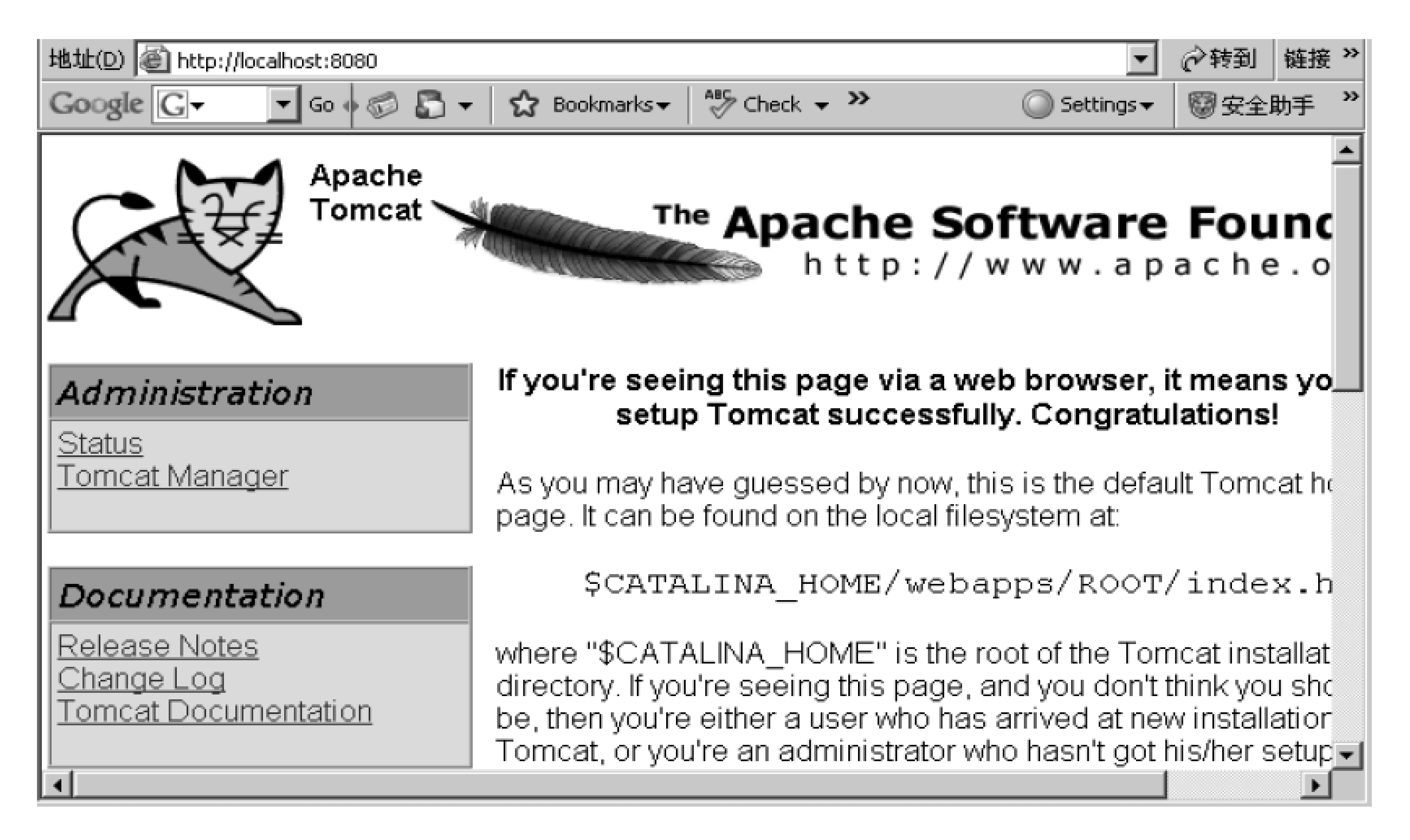


图 8-7　Tomcat 默认主页面

8.3　JSP 的基本语法

8.3.1　一个简单的 JSP 页面

下面先来看一个简单的 JSP 页面程序，该程序通过 JSP 代码实现在 Web 页面中输出一张表格。见程序清单 8-1。

程序清单 8-1：

```
<!--simple.jsp 文件代码-->
<%@page contentType="text/html; charset=GB2312"% >
<html>
<head><title>一个简单的 JSP 程序</title></head>
<body>
<center>
<table border=1>
<%
  for(int i=1;i<6;i++)
    out.println("<tr><td width=100 align=center>第 "+i+" 行</td></tr>");
%>
</table>
</center>
</body>
</html>
```

该程序的运行结果见图 8-8。

从该程序可以了解一个基本的 JSP 页面可以由 HTML 标记和 JSP 元素共同组成，其



图 8-8 程序 simple.jsp 运行结果

中 JSP 代码被标记在“<%”和“%>”之间。可以使用任何文本编辑器来编写 JSP 代码，文件编写好后以扩展名.jsp 保存。

在 JSP 页面文件中，除了 HTML 或 XML 元素外，还包含了注释元素、指令元素、动作元素和脚本元素等内容。下面就分别来介绍这些元素的使用方法。

8.3.2 JSP 的变量、方法与表达式

在 JSP 中可以通过声明来定义在代码中要使用的变量和方法，其类型可以是 Java 的基本类型或自定义类的类型。声明后的变量和方法可以在该 JSP 程序的任何地方使用。声明语句的语法格式如下：

```
<%!声明语句 1;声明语句 2;…%>
```

如：

```
<%!int i=3; %>
<%!string s1,s2;%>
<%!char getch (char c) {
        return c;
    }
%>
```

其中每条声明语句要以分号“;”作为结束标志。一个声明语句只在当前 JSP 页面有效，如果要将声明语句用于其他页面，可以将若干 JSP 页面都要用到的声明语句写成一个单独的文件，然后使用<%include %>指令或<jsp:include>动作包含到要使用的 JSP 页面中。另外也可以使用<%@ page %>包含已经声明的变量和方法，这样在页面中可以直接使用而不用重新声明。

JSP 的表达式是由变量、常量和运算符组成的式子，它可以将计算结果转换成字符串直接在页面中输出。表达式的语法格式如下：

```
<%=表达式 %>
```

如：

```
<%="Java Server Pages"%>
```

注意：

① 在 JSP 中使用表达式不需要用分号“;”作为结束标志。

② “<%=”符号间不要随意添加空格。

下面通过一个例子来帮助大家熟悉声明语句和表达式的使用，见程序清单 8-2，其运行结果见图 8-9。

程序清单 8-2：

```
<!--expression.jsp 文件代码-->
<%@page import="java.util.*"%>
<%@page contentType="text/html; charset=GB2312"%>
<head><title>声明和表达式使用示例</title></head>
<body>
<div align="center">
<%! String str;%>
<%str="现在时间是：";%>
<%! Date MyDate=new Date();%>
<font size=5 color=blue>
<b>
<%=str%><%=MyDate.toLocaleString()%>
</font>
</b>
</div>
</body>
</html>
```



图 8-9　程序 expression.jsp 运行结果

8.3.3　JSP 注释元素

JSP 中的注释可以分为两类：一类是在客户端可见的注释，可以称其为 HTML 注释；还有一类是客户端不可见的注释，这类注释有隐藏注释以及脚本代码中的注释。

1. HTML 注释

JSP 代码中的 HTML 注释可以在发回给客户端的页面文件中看到，这样可以增加最终运行结果文件的可读性。在这类注释语句中也可以使用 JSP 表达式，其语法格式如下：

```
<!--注释语句<%=表达式%>-->
```

2. 隐藏注释

这类注释语句存在于JSP程序中，但JSP引擎不会对这种注释语句进行编译，因此服务器不会将它发回给客户端，也就是说在客户端浏览器中这些注释语句是不可见的。其语法格式如下：

```
<%--注释语句--%>
```

3. 脚本代码的注释

JSP代码中也可以使用Java中的注释符号来书写注释语句，即可以使用“//”和“/*... */”来标注要添加到代码中的注释内容，这种注释同样不会被服务器发回到客户端。其语法格式为：

```
<%  //注释语句%>
```

或

```
<%/* 注释语句 */%>
```

下面通过程序清单8-3来解释这3种注释的使用，其运行结果如图8-10所示。

程序清单8-3：

```
<!--note.jsp文件代码-->
<%@page import="java.util.*"%>
<%@page contentType="text/html; charset=GB2312"%>
<html>
<head><title>JSP注释使用示例</title></head>
<body>
<!--HTML注释：当前时间为<%=(new Date()).toLocaleString()%>-->
<center>
<%--隐藏注释--%>
<br>
<%
//脚本代码注释：使用循环语句产生8行文字
for(int i=1;i<=8;i++){
    if(i%2==0)
    out.println("<font color=red>JSP注释使用示例</font><br>");
else
    out.println("JSP注释使用示例<br>");
}
%>
</center>
</body>
</html>
```

在客户端看到的运行结果如图8-10所示，浏览器所显示的页面源代码文件内容见图8-11。

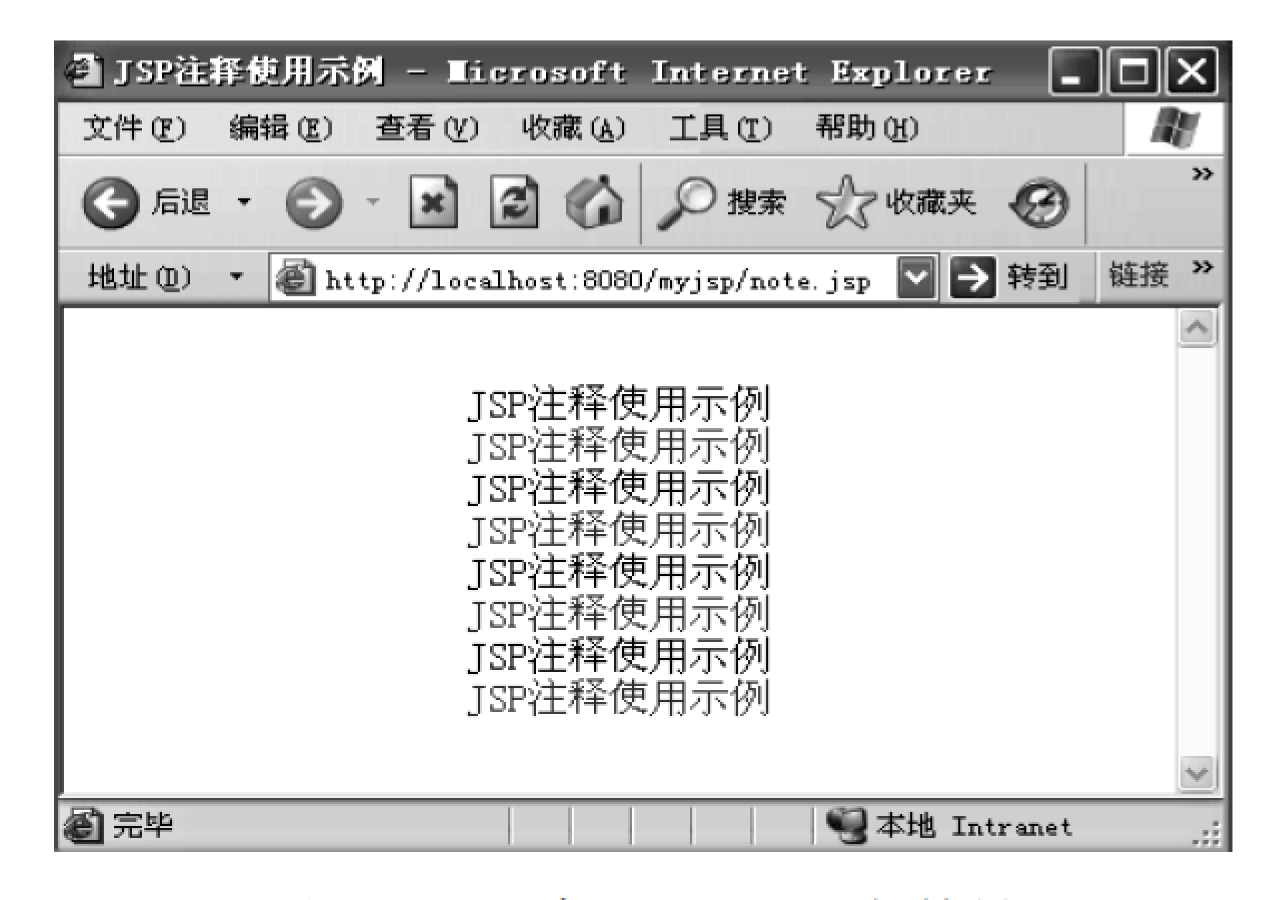
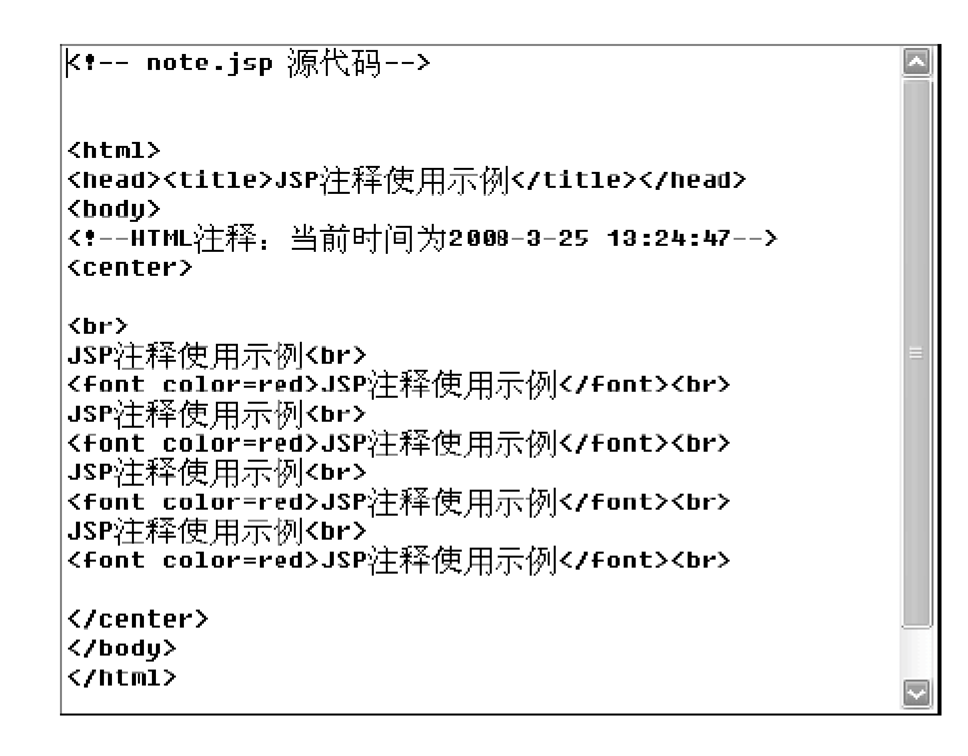


图 8-10　程序 note.jsp 运行结果



```
<!-- note.jsp 源代码-->


<html>
<head><title>JSP注释使用示例</title></head>
<body>
<!--HTML注释：当前时间为2008-3-25 13:24:47-->
<center>

<br>
JSP注释使用示例<br>
<font color=red>JSP注释使用示例</font><br>
JSP注释使用示例<br>
<font color=red>JSP注释使用示例</font><br>
JSP注释使用示例<br>
<font color=red>JSP注释使用示例</font><br>
JSP注释使用示例<br>
<font color=red>JSP注释使用示例</font><br>

</center>
</body>
</html>
```

图 8-11　客户端查看的 note.jsp 运行结果源代码

8.3.4　JSP 指令元素

JSP 指令用于通知 JSP 引擎如何处理 JSP 页面，它并不直接向客户端产生任何输出。它可以设置全局变量、声明类、方法及输入内容的类型、错误处理信息、包含的外部文件等。指令元素的语法格式如下：

```
<%@指令名 属性名 1="属性值 1", 属性名 2="属性值 2",…%>
```

常用的 JSP 指令有 include 指令、page 指令和 taglib 指令。

1. include 指令

JSP 的 include 指令用来在当前 JSP 页面中加载需要插入的文本或代码文件，这些加载的代码将和原有的 JSP 代码合并成一个新的 JSP 文件，并由 JSP 引擎编译后执行。其语法格式如下：

```
<%@include file="相对路径"%>
```

如：

```
<%@include file="head.jsp"%>
```

通过 include 加载的文件可以是 JSP 文件、HTML 文件、文本文件等，这种包含是静态包含。所谓静态包含就是指被包含的文件会插入到 JSP 文件的指定位置处。如果改变了被加载的文件的内容，则需要将包含该文件的 JSP 文件重新编译。

由于 JSP 提供的 include 指令，可以将一个复杂的 JSP 文件分成若干子文件，这样不仅可以提高 Web 程序的开发效率，更便于系统的管理和维护。

下面通过程序清单 8-4 来说明 include 指令的使用方法，其运行结果见图 8-12。其中加载的 head.jsp 文件和 showdate.jsp 文件代码分别见程序清单 8-5 和程序清单 8-6。

程序清单 8-4：

```
<!--include.jsp 文件代码-->
<%@page contentType="text/html; charset=GB2312"%>
<html>
```

```
<head><title>include指令使用示例</title></head>
<body>
<center>
<font color=blue size="7">
<%@ include file="head.jsp"%>
</font>
<hr color="#CC0000">
<h2>
<%@ include file="showdate.jsp"%>
</h2>
</center>
</body>
</html>
```



图 8-12 include.jsp 程序运行结果

程序清单 8-5：

```
<!--head.jsp文件代码-->
<%="Welcome to our web!!!"%>
```

程序清单 8-6：

```
<!--showdate.jsp文件代码-->
<%@page contentType="text/html; charset=GB2312"%>
<%@page language="Java" import="java.util.*"%>
<%
  Calendar  calendar=Calendar.getInstance();
  Date trialTime=new Date();
  calendar.setTime(trialTime);
  int year=calendar.get(Calendar.YEAR);
  int month=1+calendar.get(Calendar.MONTH);
  int day=calendar.get(Calendar.DAY_OF_MONTH);
%>
今天是 <%=year%>年 <%=month%>月 <%=day%>日
```

2. page 指令

page 指令用来定义整个 JSP 页面要使用的属性，如所使用的脚本语言、要导入的包文件、错误处理方法等。它的语法格式如下：

```
<%@ page
[language="java"]
[import="{package.class|package.* },... "]
[extends="package.class"]
[contentType="text/html;charset=ISO-8859-1" | "mimeType[;charset=CHARSET]"]
[session="True | False"]
[buffer="none | 8kb | sizekb"]
[autoFlush="Ture | False"]
[isThreadSafe="True | False"]
[info="text"]
[errorPage="relativeURL"]
[isErrorPage="True | False"]
%>
```

其中用中括号([])括起的部分表示该部分语句不是必需的。这些属性的含义见表 8-1。可以在同一个 JSP 页面中使用多个 page 指令来定义不同的属性，但除了 import 属性外，其他属性只能定义一次。

表 8-1 page 指令的属性

属　性　名	含　　义	默　认　值
language	定义该 JSP 文件要使用的脚本语言，其值目前只能设为"java"	"java"
import	定义该 JSP 文件要使用的类	空
extends	定义 JSP 文件编译时要继承的 java 父类	空
contentType	定义 JSP 文件的字符编码和输出响应的 MIME 类型	"text/html; charset=ISO-8859-1"
session	设置当前页面是否参与 HTTP 会话	True
buffer	设置到客户输出流的缓冲区大小	8kb
autoFlush	设置缓冲区填满时是否自动刷新	True
isThreadSafe	定义该 JSP 是否支持多线程	True
info	设置 JSP 页面的信息，可以通过 servlet. getServletInfo()获得	空
errorPage	定义该 JSP 页面出错时调用的页面	空
isErrorPage	设置该 JSP 页面是否为其他页面的错误处理页面	False

在程序清单 8-4 中的 include. jsp 源代码第一行和第二行可以看到 page 指令的使用方法，其中因为要在页面中显示中文字符，所以 page 指令的 charset 值设为 GB2312。

3. taglib 指令

taglib 指令用于用户自定义标签库以及标记的前缀，其语法格式如下：

```
<%@ taglib uri="taglibURI" prifex="tagPrefix" %>
```

其中,属性 uri 用来描述如何查找标签库和标签描述文件,属性 prifex 定义了在 JSP 文件中要使用这个标签的前缀,其值不可以设置为 jsp、jspx、java、javax、sun、servlet 和 sunw。随着 JSP 技术的不断发展,其标签库不断增强,在 JSP 2.0 中增加了 JSTL 标签库。

8.3.5 JSP 动作元素

JSP 动作元素采用 XML 格式书写,在请求处理时起作用,用于控制 JSP 引擎的行为。常用的 JSP 动作包括＜jsp：param＞、＜jsp：include＞、＜jsp：useBean＞、＜jsp：setProperty＞、＜jsp：getProperty＞、＜jsp：forward＞和＜jsp：plugin＞等。

1. ＜jsp：param＞动作

＜jsp：param＞动作用来为 JSP 页面的其他标记提供附加信息,通常它以"名/值"的方式传递到页面中,其语法格式如下:

```
<jsp:param name="参数名" value="参数值"/>
```

其中属性 name 用于设置参数名称,属性 value 用于设置参数值。

2. ＜jsp：include＞动作

＜jsp：include＞动作用来在生成的 Web 页面中插入指定的文件,其语法格式如下:

```
<jsp:include page="文件名" flush="true"/>
```

或

```
<jsp:include page="文件名" flush="true">
<jsp:param name="参数名 1" value="参数值 1"/>
<jsp:param name="参数名 2" value="参数值 2"/>
⋮
</jsp:include>
```

其中属性 page 的值为要插入的文件的相对路径,如果该文件和包含它的 JSP 文件在同一目录下,则 page 值设为要插入的文件名。＜jsp：param＞动作用于向 JSP 页面传递若干个参数,我们可以通过 request.getParameter("参数名")获取传递到页面的参数值。

＜jsp：include＞动作和之前介绍过的 include 指令不同,include 指令是静态包含,它是在 JSP 文件被转换为 Servlet 时插入文件,包含的内容是固定不变的。而＜jsp：include＞动作是在页面文件执行时才将文件插入,如果被插入的文件修改过,＜jsp：include＞动作可以得知并对文件重新编译。＜jsp：include＞动作在插入文件时,JSP 文件和被插入的文件在逻辑上是独立的,它允许包含静态文件和动态文件。如果包含的文件是静态文件,那么＜jsp：include＞执行的操作就是把文件内容直接加载到 JSP 文件的指定位置,该文件内容并不会被 JSP 引擎执行;如果包含的文件是动态文件,该文件就会被 JSP 引擎执行。

下面通过程序清单 8-7 来说明如何使用＜jsp：include＞动作,其运行结果见图 8-13。其中加载的 welcome.jsp 文件代码见程序清单 8-8。

程序清单 8-7:

```
<!--include_action.jsp 文件代码-->
```

```
<%@  page contentType="text/html; charset=gb2312" language="java" %>
<html>
<body>
<center>
<h3>include动作示例</h3>
<hr />
<jsp:include page="welcome.jsp" flush="true">
<jsp:param name="yourname" value="Emma" />
</jsp:include>
</center>
</body>
</html>
```

程序清单 8-8：

```
<!--welcome.jsp文件代码-->
<%@  page contentType="text/html; charset=gb2312" language="java" %>
<br>
<%=request.getParameter("yourname")%>,您好！
<br>
现在是 <%=new java.util.Date().toLocaleString()%>
<br>
欢迎您访问本网页。
```

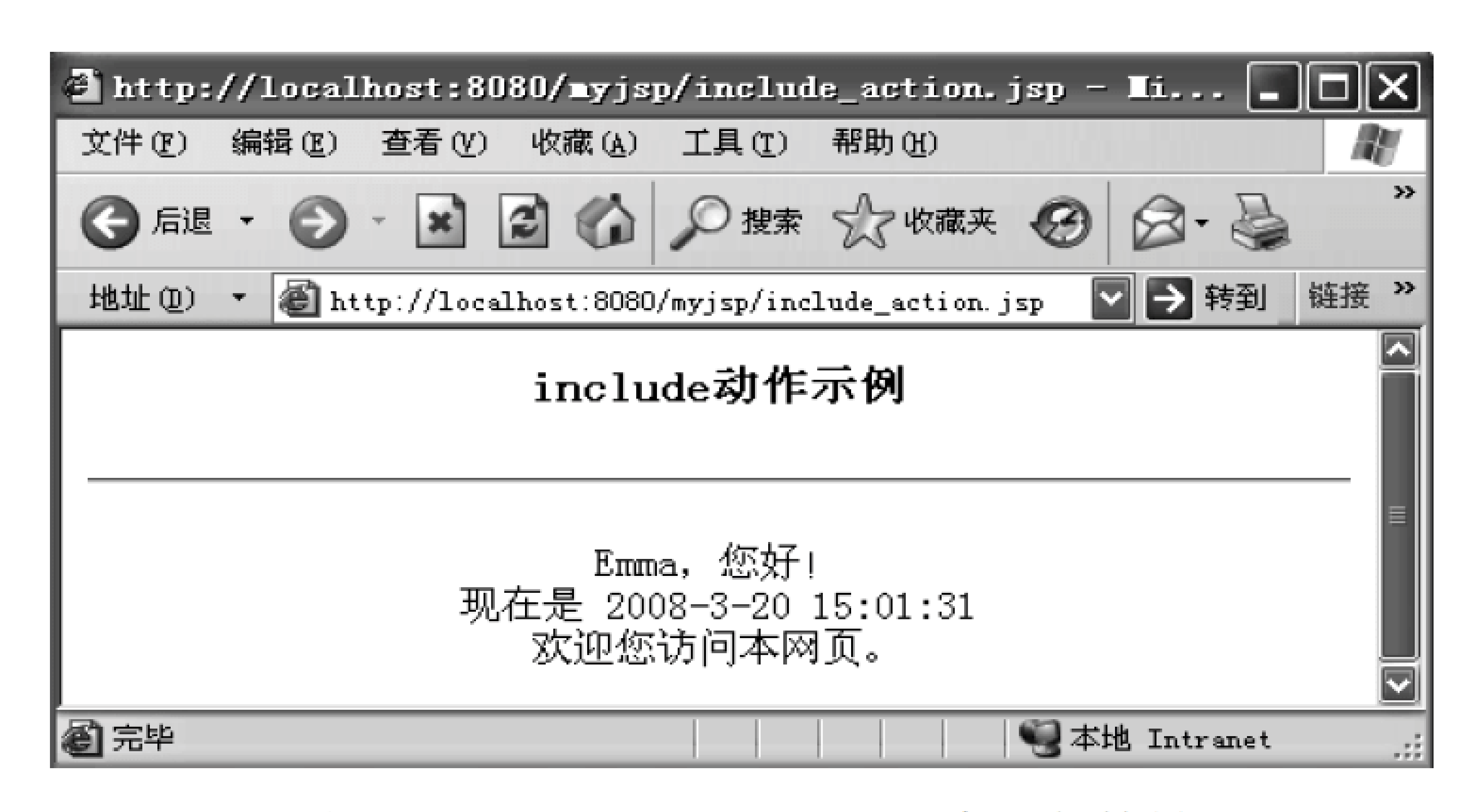


图 8-13　include_action.jsp 程序运行结果

3. ＜jsp：setProperty ＞动作、＜jsp：getProperty＞动作和＜jsp：useBean＞动作

（1）＜jsp：setProperty ＞动作。＜jsp：setProperty ＞动作是和＜jsp：useBean＞动作一起使用的，用来设置 Bean 的属性值，其语法格式如下：

```
<jsp:setProperty name="Bean的名字" property="*"/>
```

或

```
<jsp:setProperty name="Bean的名字" property="属性名" [param="参数名"]/>
```

或

```
<jsp: setProperty name="Bean的名字" property="属性名" value="属性值"/>
```

其中字段含义如下。

① name 是已经由<jsp：useBean>动作引入的 Bean 的实例名。

② property＝"＊"用于存储用户在 JSP 页面输入的所有数据以匹配 Bean 中定义的属性，是一种设置 Bean 属性的快捷方式，当使用 property＝"＊"语句时，Bean 中定义的属性就不需要和 HTML 表单中的顺序一致。

③ property＝"属性名" [param＝"参数名"]用于使用 request 中的一个参数值来指定 Bean 中定义的一个属性值，property 属性的值设定 Bean 的属性名称，该属性名应和 request 参数的名字相同，如果不相同，则可以使用 param＝"参数名"指定 request 中的参数名，若 request 中的参数有空值，那么对于的 Bean 属性值也不会设置任何值。

④ property＝"属性名" value＝"属性值"通过指定 value 属性的值来设置 Bean 的属性值，该值可以是字符串或表达式，如果为表达式，其类型应和要设定的 Bean 属性的类型一致，如果是字符串，则 JSP 引擎会使用 valueof()方法将该字符串值自动转换为要设定的 Bean 属性的类型。

（2）<jsp：getProperty >动作。<jsp：getProperty >动作也是和<jsp：useBean>动作一起使用，用来获取 Bean 中属性的值，所得到的值会转换成相应的字符串，然后发送到输出流输出。其语法格式如下：

```
<jsp: getProperty name="Bean的名字" property="属性名"/>
```

其中，属性 name 指定已经由<jsp：useBean>动作引入的 Bean 的实例名，属性 property 用来设置要获取的 Bean 中的属性名称。

（3）<jsp：useBean>动作。<jsp：useBean>动作用于加载要在 JSP 页面中使用的已经定义的 JavaBean，使用该动作在 JSP 页面中创建该 Bean 的实例，并指定它的名字以及作用范围。其语法格式如下：

```
<jsp: useBean id="name" scope="page|request|session|application" class="classname"/>
```

或

```
<jsp: useBean id=" name " scope="page|request|session|application "
class="classname" type="typename"/>
```

或

```
<jsp: useBean id=" name " scope="page|request|session|application "
beanName="beanName" type="typename"/>
```

或

```
<jsp: useBean id=" name " scope="page|request|session|application " type="typename"/>
```

其中字段含义如下：

① id＝" name "定义要加载的 Bean 的实例的名字，使用该名字来区分页面中使用的不同 JavaBean，该名字区分大小写。若要使用在其他<jsp：useBean>中创建的 Bean，那么该

id 值应该与那个 id 值一致。

② scope="page|request|session|application "指定 Bean 存在的范围和 id 属性值的有效范围,其默认值为 page。

"page"描述了可以在当前 JSP 文件及该文件的所有静态包含文件中使用 Bean,直到页面执行完毕或用户离开该页面时,Bean 不可用。

"request"描述了可以在 request 对象存在期间使用 Bean。

"session"描述了可以在 session 对象存在期间使用 Bean。

"application"描述了可以在相同的 application 对象使用期间使用一个 Bean。

③ class="classname"指定要加载的 Bean 的类文件路径。这个 class 不能是抽象的,其中若定义构造函数,该函数应该是一个公用的不带参数的构造函数。

下面通过程序清单 8-9 的 simplecart.html 和程序清单 8-10 的 simplecart.jsp 文件来说明如何使用<jsp:setProperty>动作和<jsp:useBean>动作获取页面提交的数据信息,其中使用的 Java Bean 程序代码见程序清单 8-11。

程序清单 8-9:

```
<!--simplecart.html 文件代码-->
<html>
<head><title>simple cart example</title></head>
<body>
<form method="post" action="simplecart.jsp">
<br>
  <table align="center">
    <tr>
      <td colspan="2">
      <font color="red">请选择需要购买的书籍,并填写数量</font>
      </td>
      </tr>
      <tr>
        <td>目前可供选择的书籍有:</td>
        <td>
        <select name="item">
          <option>哈里波特系列
          <option>灰姑娘
          <option>三国演义
          <option>红楼梦
          <option>十万个为什么
        </select>
        </td>
      </tr>
      <tr>
        <td align="right">数量:</td>
        <td><input type="text" name="number" size=15 /></td>
      </tr>
```

```
        <tr>
          <td align="right"><input type=submit name="submit" value="提交"></td>
          <td><input type=reset name="reset" value="重置"></td>
      </tr>
    </table>
</form>
</body>
</html>
```

程序清单 8-10：

```
<!--simplecart.jsp 文件代码-->
<%@ page contentType="text/html; charset=GB2312"%>
<%
   request.setCharacterEncoding("gb2312");
%>
<html>
<jsp:useBean id="cart" scope="page" class="myBean.simplecart" />
<jsp:setProperty name="cart" property="*" />
<body>
<br><font size="5" face="黑体">您选购的商品为：</font>
<br>
<hr>
<table>
   <tr>
      <td width="200">书名</td>
      <td width="200">数量</td>
      </tr>
      <%
         String item=cart.getItem();
      int number=cart.getNumber();
      out.println("<tr><td width=200>《"+item+"》</td>");
      out.println("<td width=200>"+number+"</td></tr>");
   %>
</table>
</body>
</html>
```

程序清单 8-11：

```
//simplecart.java 文件代码
package myBean;
public class simplecart {
      public String item;
      public int number;
      public void setItem(String name) {
```

```
            item= name;
        }
        public String getItem() {
            return item;
        }
        public void setNumber(int number)      {
            this.number= number;
        }
        public int getNumber() {
            return this.number;
        }
    }
```

当运行 simplecart.html 文件时，可以看到如图 8-14 所示页面，按要求在下拉框中选择书籍名称并在文本框中填写数量后，单击“提交”按钮，会将该页面表单中的数据提交到 simplecart.jsp 程序并显示如图 8-15 所示结果。



图 8-14　simplecart.html 程序运行结果

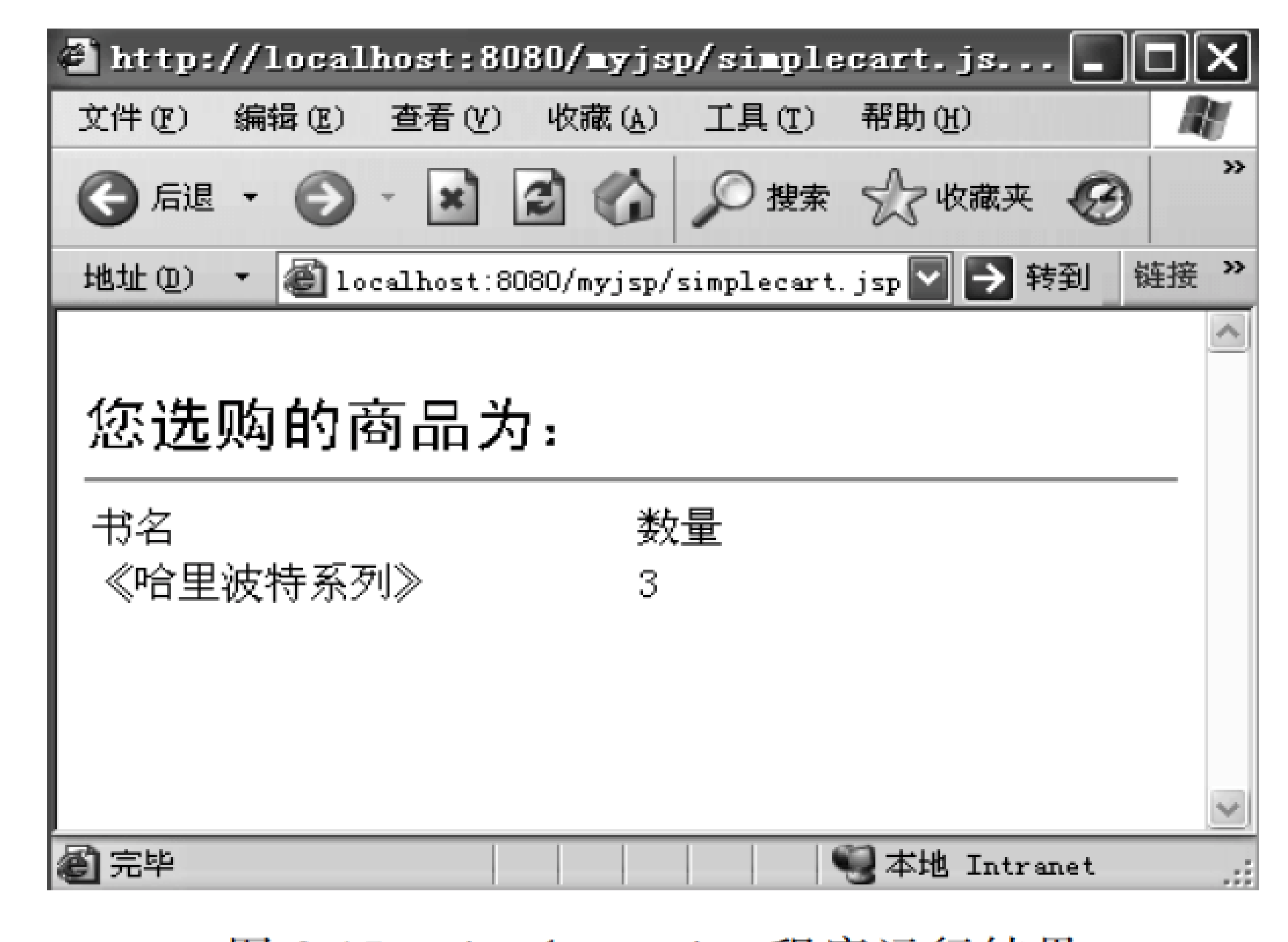


图 8-15　simplecart.jsp 程序运行结果

4. <jsp：forward>动作

<jsp：forward>动作用于停止当前页面的剩余操作，而转向另一个 HTML 或 JSP 页面文件。客户端看到的仍然是当前页面的地址，但显示内容为转向的新页面。其语法格式如下：

```
<jsp:forward page="目标文件 URL" />
```

或

```
<jsp:forward page="目标文件 URL">
<jsp:param name="参数名 1" value="参数值 1"/>
<jsp:param name="参数名 2" value="参数值 2"/>
⋮
</jsp:forward>
```

其中属性 page 设置了要转向的目标文件的文件名或 URL。同时可以将<jsp：param>和<jsp：forward>结合使用，用于向动态文件传递若干参数信息。我们通常会在登录页面中使用<jsp：forward>动作，当验证登录用户的信息时，若通过验证则使用<jsp：forward>动作转向登录成功的页面，若未通过身份验证则使用<jsp：forward>动作转向重新登录的页面。

下面通过程序清单 8-12 来说明如何使用<jsp：forward>动作，其中 check.jsp 文件代码见程序清单 8-13，error.jsp 文件代码见程序清单 8-14。运行 login.jsp 文件后可以看到如图 8-16 所示页面。如果在页面内输入姓名“Emma”和密码“123456”，提交后页面会通过<jsp：forward>动作转向 welcome.jsp 页面(welcome.jsp 程序源代码见程序清单 8-8)，如图 8-17 所示；否则页面转向 error.html 提示输入错误(如图 8-18 所示)，并在该页面打开 3 秒后重新转到 login.jsp 页面。

程序清单 8-12：

```
<!--login.jsp文件代码-->
<%@ page contentType="text/html; charset=gb2312" %>
<html>
<body>
<form method="post" action="check.jsp">
<table>
  <tr>
    <td>用户名:</td>
    <td><input type="text" name="name"></td>
  </tr>
  <tr>
    <td>密码:</td>
    <td><input type="password" name="password"></td>
  </tr>
  <tr>
    <td align="center"><input type="submit" value="提交"></td>
    <td align="center"><input type="reset" value="重置"></td>
  </tr>
</table>
```

```
</body>
</html>
```

程序清单 8-13：

```
<!--check.jsp文件代码-->
<%@ page contentType="text/html; charset=gb2312" %>
<html>
<body>
<%
    String name=request.getParameter("name");
    String password=request.getParameter("password");
    if(name.equals("Emma")&&password.equals("123456"))
    {
%>
<jsp:forward page="welcome.jsp">
    <jsp:param name="yourname" value="<%=name%>"/>
</jsp:forward>
<%}else{%>
<jsp:forward page="error.html"/>
<%}%>
</body>
</html>
```

程序清单 8-14：

```
<!--error.jsp文件代码-->
<html>
<head>
<meta http-equiv="refresh" content="3;url=login.jsp">
<body>
<center>
<br><br>
<font color="#CC0000" size="7">输入的用户名或密码不正确,请重新输入!</font>
</body>
</html>
```

图 8-16　login.jsp 运行结果



图 8-17　输入信息正确

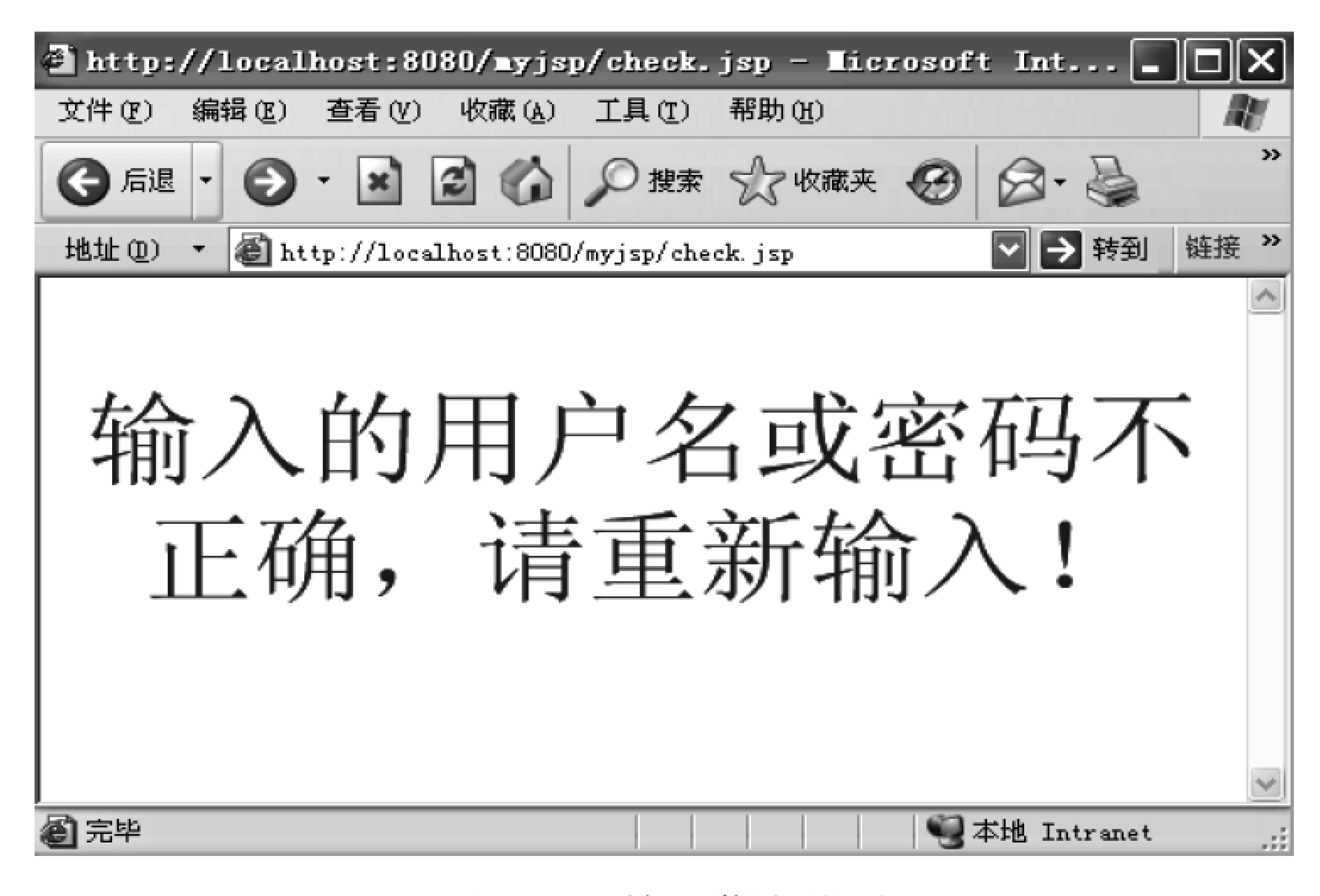


图 8-18　输入信息错误

5. <jsp:plugin>动作

<jsp:plugin>动作用于在客户端浏览器插入 Applet 程序或 JavaBean。当 JSP 文件执行后发送到客户端时，<jsp:plugin>动作会产生<object>或<embed>标记以使浏览器运行 Applet 或 JavaBean。其语法格式如下：

```
<jsp:plugin type="bean | applet"
            code="classFileName"
            codebase="classFileDirectoryName"
            name="instanceName"
            [archive="URIToAchive,…"]
            [align="bottom | top| middle| left| right"]
            [height="displayPixels"]
            [width="displayPixels"]
            [hspace="leftrightPixels"]
            [vspace="topbottomPixels"]
            [jreversion="JREVersionNumber | 1.1"]
            [nspluginurl="URLToPlugin"]
            [iepluginurl="URLToPlugin"]>
```

```
[<jsp: params>
[<jsp: param name="parameterName"
        value="parameterValue| <%=expression%>"/>]
</jsp: params>]
[<jsp: fallback>text message for user</jsp: fallback>]
</jsp: plugin>
```

其中字段含义如下。

(1) 属性 type 设置要执行的插件的类型,必须为 Bean 和 Applet 其中之一。

(2) 属性 code 设置要运行的类文件的名称,该名称必须包含扩展名。

(3) 属性 codebase 设置要运行的类文件的路径,默认值为该 JSP 文件的路径。

(4) 属性 name 设置要运行的类文件的实例名。

(5) 属性 archive 设置用于 codebase 属性指定的目录下的类文件的相关文件路径,路径间以逗号(,)分隔。

(6) 属性 align 设置插件的排列方式。

(7) 属性 height 和 width 设置插件要显示的长和宽,该属性值为数值,单位为像素。

(8) 属性 hspace 和 vspace 设置插件显示时屏幕的左右、上下预留空间的大小,该属性的值为数值,单位为像素。

(9) 属性 jreversion 设置插件运行的 JRE 版本,默认值为 1.1。

(10) 属性 nspluginurl 设置浏览器为 Netscape Explore 的用户能够下载 JRE 的下载地址。

(11) 属性 iepluginurl 设置浏览器为 Internet Explore 的用户能够下载 JRE 的下载地址。

(12) 语句<jsp: params> <jsp: param name="parameterName" value="parameterValue| <%=expression%>"/></jsp: params>描述了要向插件传递的参数信息。

(13) 语句<jsp: fallback>text message for user</jsp: fallback>描述了插件不能运行时在页面上显示给用户的文字信息。

下面通过程序清单 8-15 来说明如何使用<jsp: plugin>动作,其运行结果见图 8-19。

程序清单 8-15:

```
<!--plugin.jsp 文件代码-->
<%@ page contentType="text/html; charset=gb2312" %>
<html>
<head><title>plugin 动作使用示例</title></head>
<body>
<center>
<br>
<font size="6">jsp: plugin 动作使用示例</font><br>
<hr>
<!--用 plugin 加载 Applet-->
```

```
<jsp: plugin type= "applet" code= "DrawingGraphic.class" height= "300" width= "300">
    <jsp: fallback>
        Plugin tag OBJECT or EMBED not supported by browser.
    </jsp: fallback>
</jsp: plugin>
</center>
</body>
</html>
```

其加载的类文件 DrawingGraphic 的 Java 程序代码见程序清单 8-16，该文件编译后的 Class 文件 DrawingGraphic. class 和 plugin. jsp 文件放在同一目录下。

程序清单 8-16：

```
//DrawingGraphic.java 文件代码
import java.applet.* ;
import java.awt.* ;
public class DrawingGraphic extends Applet {
    int width, height;
    public void init() {
        width= getSize().width;
        height= getSize().height;
        setBackground( Color.pink );
    }
    public void paint( Graphics g ) {
        g.setColor(Color.red);
        g.drawRect(10,30,getWidth()/2- 50,getHeight()/2- 50);
        g.drawOval(10,30,getWidth()/2- 50,getHeight()/2- 50);
        g.drawLine(10,30,5+ getWidth()/2- 50,30+ getHeight()/2- 50);
        g.setColor( Color.blue );
        g.fillRoundRect( 100, 160, 80, 110, 30, 40 );
        g.setColor( Color.darkGray );
        g.fillArc( 140, 40, 120, 120, 90, 135 );
    }
}
```

8.3.6 JSP 脚本元素

JSP 脚本（Scriptlet）是编写 JSP 代码时使用很频繁的元素，它通常是用 Java 编写的脚本代码，可以定义变量和函数，也可以进行表达式求值、产生输出等操作。

Scriptlet 是介于“<%”和“%>”之间的 Java 代码段，可以在服务器端被编译执行，执行结果会嵌入页面文件并由服务器发回给客户端浏览器。

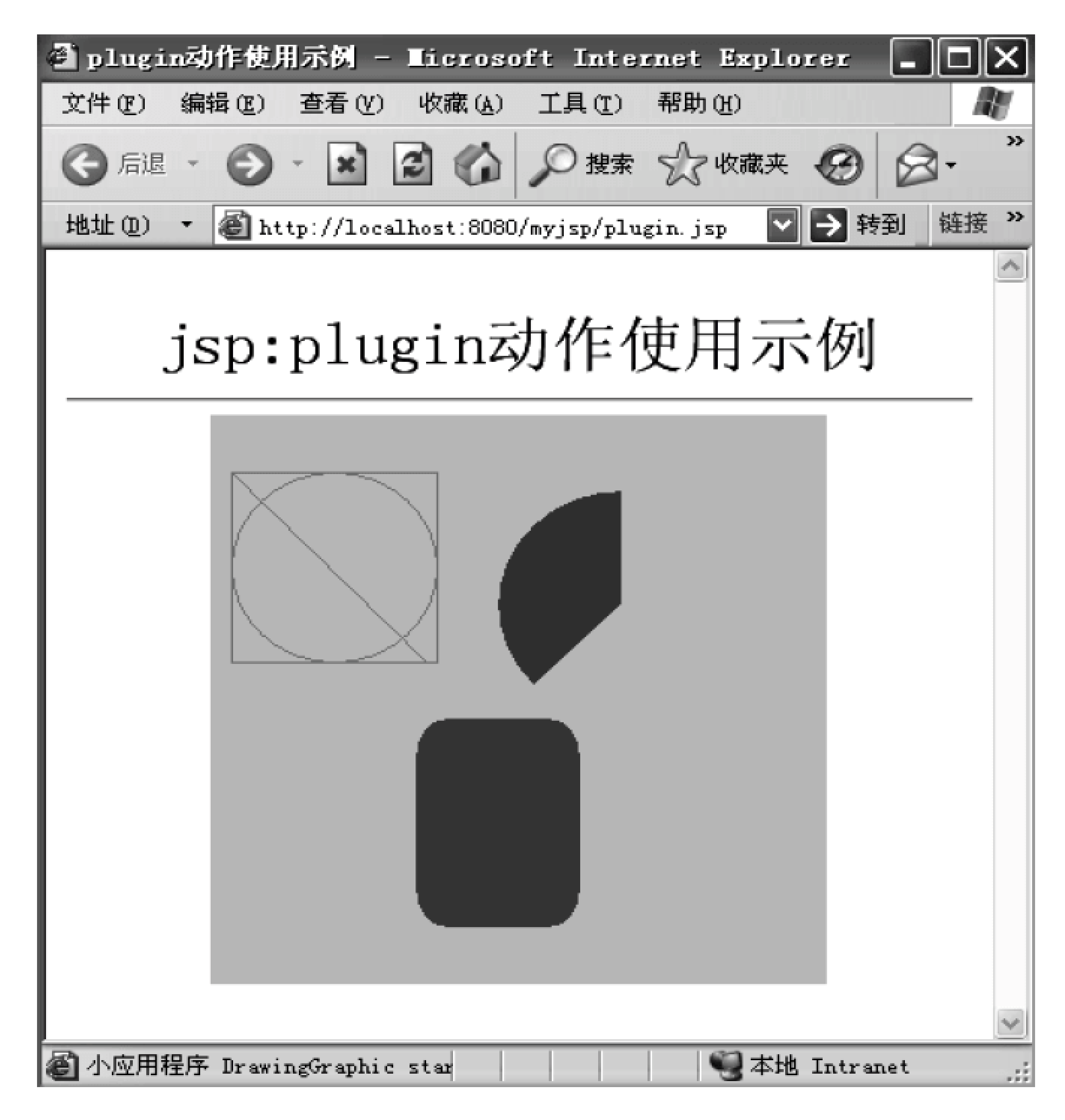


图 8-19 plugin.jsp 程序运行结果

小　　结

本章介绍了 JSP 的基本知识，主要包括：JSP 的工作原理；JSP 运行环境的安装和设置；JSP 的脚本、变量和方法的声明和使用；JSP 的注释方式；JSP 的指令元素：include、page 和 taglib 指令的功能和使用方法；JSP 的常用的动作元素：<jsp：param>、<jsp：include>、<jsp：useBean>、<jsp：setProperty>、<jsp：getProperty>、<jsp：forward>和<jsp：plugin >的功能和使用方法。

练　习　8

1. 填空题

（1）JSP 是一种运行于________端的 Web 程序开发技术。

（2）JSP 指令用于通知 JSP 引擎如何处理 JSP 页面，它并不直接向客户端产生任何输出。常用的 JSP 指令有________指令、________指令和________指令。

（3）include 指令的作用是________。

（4）<jsp：useBean>动作用于加载要在 JSP 页面中使用的已经定义的________。

（5）<jsp：forward>动作用于停止当前页面的剩余操作，而转向另一个 HTML 或 JSP 页面文件。客户端看到 URL 地址为________，但显示内容为________。

2. 选择题

（1）JSP 程序首次运行时，服务器会先将该 JSP 源文件翻译成一个________文件，然后

进行编译再由服务器运行。

A. HTML　　B. Servlet　　C. Java Bean　　D. JavaScript

(2) 可以在同一个 JSP 页面中使用多个 page 指令来定义不同的属性，但除了________属性外，其他属性只能定义一次。

A. contentType　　B. session　　C. import　　D. language

(3) 当使用语句<jsp:param name="paramname" value="paramvalue" >来向其他目标 JSP 页面传递参数信息时，可以在目标页面中使用________语句获取所设定的参数值。

A. response.getParameter("paramname")

B. request.getAttribute("paramname")

C. request.setAttribute("paramname")

D. request.getParameter("paramname")

3. 实验题

编写 JSP 程序实现用户注册页面，在该页面用户需填写相关个人信息，单击“提交”按钮后在目标页面将所有信息显示出来。

第 9 章　JSP 的内置对象

9.1　内置对象概述

JSP 开发动态网页常要创建并使用多种对象。内置对象(Implicit Objects)是一类特殊的对象,它们是 JSP 文件中不需要声明就可以使用的对象,也称它们为隐含对象。这些对象包含某种特定的信息,如 HTTP 请求、响应等,通过它们可以帮助用户获取并使用这些信息。由于内置对象是通过 JSP 容器创建和管理,用户无须考虑这些内置对象是如何产生就可直接使用。从这一点来说,内置对象的出现,简化 JSP 页面的开发,提高了开发的便利性。

JSP 的内置对象有 9 种(见表 9-1):out 对象、request 对象、response 对象、session 对象、application 对象、exception 对象、config 对象、page 对象和 pageContext 对象。下面将详细依次介绍这 9 种内置对象。

表 9-1　JSP 的内置对象

对　象	类　　别	描　　述
out	javax. servlet. jsp. JspWriter	一个输出流对象
request	javax. servlet. ServletRequest 的子类	触发 JSP 文件的请求对象
response	javax. servlet. ServletResponse 的子类	返回给客户户的响应对象
session	javax. servlet. http. HttpSession	用户的会话对象
application	javax. servlet. ServletContext	JSP 页面的应用上下文对象
pageContext	javax. servlet. jsp. PageContext	JSP 页面的上下文对象
config	javax. servlet. ServletConfig	初始化 JSP Servlet 的对象
page	java. lang. Object	JSP 页面 servlet 的当前请求处理实例
exception	java. lang. Throwable	访问错误页面产生的异常对象

9.2　out 对象

out 对象是 javax. servlet. jsp. JspWriter 的一个对象实例,表示一个输出流,是通过 out 对象向客户端发送信息,在 JSP 页面中由 pageContext 对象(见第 9. 9 节)的相关方法自动创建的"javax. servlet. jsp. JspWriter"的实例。out 对象的作用范围为 page,即只在本 JSP 页面有效。为了实现客户访问 Servlet 输出流的相关数据,可以根据要求调用 out 对象的方法(见表 9-2)。

表 9-2　out 对象的常见方法

方　　法	说　　明
void clear()	清除缓冲区的内容
void clearBuffer()	清除缓冲区的当前内容
void close()	关闭输出流
void flush()	强制输出缓冲区的数据
boolean isAutoFlush()	判断缓冲区是否具有强制输出 autoFlush 的功能,有返回 true,否则返回 false
int getBufferSize()	返回缓冲区的大小
int getRemaining()	返回缓冲区中没有占用的空间
void newLine()	输出一新行
void print()	有一个参数,参数可以是 boolean,char,char[],double,float,int,long,Object,String 中的任何一种。该方法输出信息
void println()	有一个参数,参数类型同 print()。该方法实现换行输出信息

【例 9-1】 out 对象实现表格输出。见程序清单 9-1,运行结果如图 9-1 所示。

程序清单 9-1:

```
<!--   JSP9-1.jsp   -->
<%@page contentType="text/html; charset=gb2312" language="java"
      buffer="5kb" autoFlush="false"%>
<html>
<head>
<title>out 对象的简单应用</title>
</head>

<body>
<%
out.println("<table border='1'>");
out.println("<tr><th bgcolor='#FF6600'>类型</td><th bgcolor='#FF6600'>输出内容</td> </tr>");
out.println("<tr><td>逻辑值</td><td>");
out.print(true);                  //输出 true
out.println("</td></tr>");
out.println("<tr><td>字符值</td><td>");
out.println('c');                 //输出字符
out.println("</td></tr>");
out.println("<tr><td>整数值</td><td>");
out.println(34);                  //输出整数
out.println("</td></tr>");
out.println("<tr><td>长整数值</td><td>");
out.println(32);                  //输出长整数
out.println("</td></tr>");
out.println("<tr><td>字符串</td><td>");
out.println("使用 out 对象输出字符串");  //输出字符串
```

```
out.println("</td></tr>");
out.println("<tr><td>对象值</td><td>");
out.println(new Object());                    //输出对象
out.println("</td></tr>");
out.println("<tr><td>缓冲区的大小</td><td>"+out.getBufferSize()+"</td></tr>");
out.println("<tr><td>缓冲区的剩余大小</td><td>"+out.getRemaining()+"</td></tr>");
out.println("<tr><td>autoFluash 的状态</td><td>"+out.isAutoFlush()+"</td></tr>");
out.println("</table><hr>");
out.println("<br>输出：准备关闭");
out.close();
out.println("<br>关闭缓冲区后输出");      //输出无效
%>
</body>
</html>
```



图 9-1　out 对象应用实例的运行结果

在上个例子中，通过 page 指令设置整个 JSP 页面中缓冲区的大小为 5Kb（默认为 8Kb），autoFlush 设置为 false（默认为 true），不能自动输出缓冲区。通过 out 对象将输出流的这些数据输出。另外有一点要注意：out 对象的 println()方法虽然具有换行的作用，但是这个换行的处理会被浏览器忽略掉，不会显示。所以要输出
实现换行处理。

9.3　request 对象

在了解 request 对象之前，首先回顾万维网（World Wide Web）的运行。万维网的运行与 HTTP 通信协议相关。用户通过客户端的浏览器请求 Web 服务器的服务，并向 Web 服务器发送一个 HTTP 请求。Web 服务器接收到请求，并对此进行处理，并向客户端发回

HTTP 响应。在 JSP 中，在一次 HTTP 请求中，用内置对象 request 封装由客户传递给 Web 服务器的数据。

request 对象取决于 javax. servlet. ServletRequest 类的子类 javax. servlet. http. HttpServletRequest。request 对象可以获取客户端浏览器的 header 头请求、Cookies 信息段请求和 session 会话请求。要获取客户发送的具体信息，需调用 request 对象的相应方法来实现。request 对象的常见方法见表 9-3。

表 9-3　request 对象的常见方法

方　　法	说　　明
Object getAttribute(String)	获取请求指定属性名的值
Enumeration getAttributeNames()	返回 request 对象包含的属性名
void setAttribute(String, Object)	设置指定属性的值
String getContentLength()	获取用户提交信息的整个长度
Cookie[] getCookies()	获取请求的信息段
String getParameter(String)	获取请求指定参数的值
Enumeration getParameterNames()	获取请求中参数的名称
String[] getParameterValues(String)	返回特定参数对应的值
String getHeader(String)	获取请求的头信息
Enumeration getHeaderNames()	获取请求的头名字的一个枚举
String getMethod()	获取请求的 HTTP 方法（如 get、post 和 put）
String getPathInfo()	获取请求的 URI 的 servlet 路径
String getProtocol()	返回请求中的协议和版本
String getQueryString()	获取请求的查询字符串，用于用户以 get 方法发送
String getRomoteUser()	获取创建请求对象用户的名字
String getRomoteHost()	返回请求代理的服务器名
String getRomoteAddr()	返回接收请求的服务器的 IP 地址
String getRequestedSessionId()	获取请求的特定会话编号
String getServerName()	返回接收 request 请求的服务器名
int getServetPost()	返回接收 request 请求的服务器端口号
String getSevletPath()	获取对应触发 servlet 的 request 对象 URI
HttpSession getSession([boolean])	获取 request 对象相连的合法 session。布尔参数是一个可选项，如果参数为 true 或无参数，则为请求创建一个新的会话 session

request 对象非常重要。request 对象结合其他内置对象可以实现获取客户提交的信息、对相应的数据进行分析处理、实现会话管理等方面发挥重要的作用。在例 9-2 中将对 request 对象的作用做一个初步了解。

【例 9-2】 一个 request 对象的简单应用，获取用户提交的信息。具体代码见程序清单 9-2 和程序清单 9-3。其中，程序清单 9-2 定义了一个表单，将表单数据提交给程序清单 9-3，运行结果如图 9-2 和图 9-3 所示。

程序清单 9-2：

```
<!--    JSP9-2.jsp    -->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<title>无标题文档</title>
</head>

<body>
<p>request 对象的常见方法的应用。</p>
<form action="JSP9-3.jsp" method="get">    <!--表单提交到 JSP-3.jsp-->
用户名：<input type="text" name="username"/><br/>
密码：  <input type="password" name="password"/><br/>
<input type="submit" value="发送"/>
</form>
</body>
</html>
```



图 9-2 显示表单界面

程序清单 9-3：

```
<!--    JSP9-3.jsp    -->
<%@  page contentType="text/html; charset=gb2312" language="java"
import="java.util.*" %>
<html>
<head>
<title>request 对象的常见方法的应用</title>
```

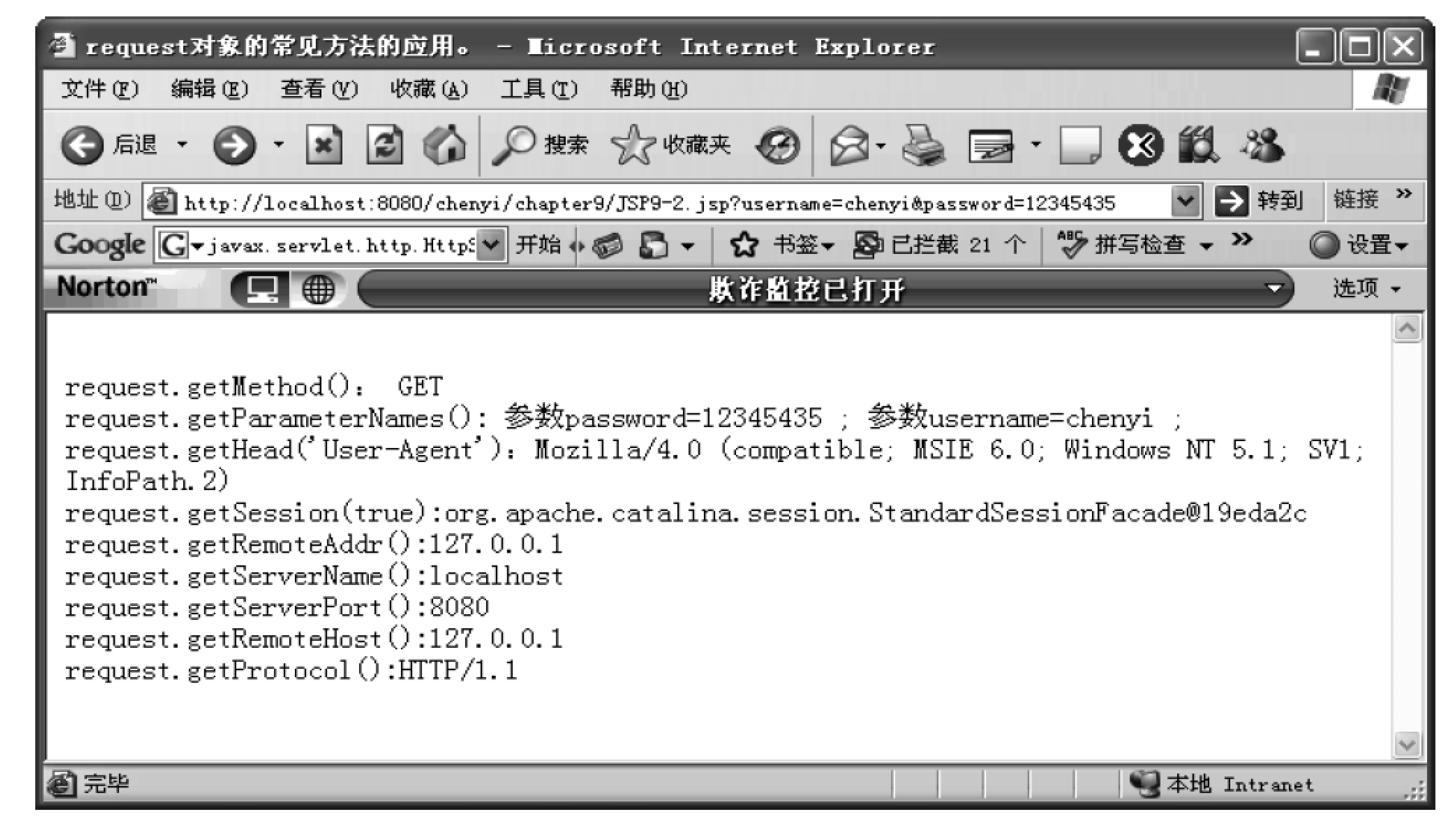


图 9-3 输出请求信息

```
</head>

<body>
<%
out.println("<br> request.getMethod(): ");
out.println(request.getMethod());              //返回请求的方法;

out.println("<br> request.getParameterNames(): ");
Enumeration enumdata= request.getParameterNames();
                                               //返回参数名的枚举;
while(enumdata.hasMoreElements()){
   String s= (String)enumdata.nextElement();   //返回参数名
   out.println("参数"+ s+ "= "+ request.getParameter(s)+ " ;");
                                               //输出参数名和对应的参数值
}

out.println("<br> request.getHead('User- Agent'): ");
out.println(request.getHeader("User- Agent"));
                                               //输出头文件中的 User- Agent 的值
out.println("<br> request.getSession(true): "+ request.getSession(true));
                                               //返回 session 编号
out.println("<br> request.getRemoteAddr(): "+ request.getRemoteAddr());
                                               //返回服务器的地址
out.println("<br> request.getServerName(): "+ request.getServerName());
                                               //返回服务器的名字
out.println("<br> request.getServerPort(): "+ request.getServerPort());
                                               //返回服务器的端口
```

```
out.println("<br> request.getRemoteHost(): "+ request.getRemoteHost());
                                                    //返回客户机名
out.println("<br> request.getProtocol(): "+ request.getProtocol());
                                                    //返回通信协议
%>
</body>
</html>
```

程序清单 9-2 将表单数据用“GET”方式发送 JSP9-3.jsp。JSP9-3.jsp 接收数据，并利用 request 对象来获取请求的 HTTP 方法，头名字为 User-Agent 的信息，以及接收数据端的 IP 地址、端口以及服务器名等信息。值得注意的是，request 对象仅在本请求内有效。

9.4 response 对象

9.4.1 response 对象的概述

response 对象表示 HTTP 响应数据，它可以将服务器处理用户请求 request 的结果返回给用户。JSP 中将这些服务器响应用户请求的数据封装成为一个 response 对象。从本质上来说，response 对象是实现 javax.servlet.ServletResponse 接口的实例，是由 JSP 容器生成。可对客户的请求做出动态的响应：向客户端发送数据(如 HTTP 文件头信息)、实现客户端转向其他资源、定时刷新、与 Cookie 相关的操作(如刷新、保存时间操作等)。response 的作用域为 page，表示对当前页面有效。response 中定义的常见方法见表 9-4。

表 9-4 response 对象的常见方法

方　法	说　明
void addCookie(Cookie)	添加一个 Cookie 对象，用来保存客户端的用户信息
void addHeader(String,String)	添加一个指定 HTTP 头名以及对应的值
boolean containsHeader(String)	检查响应信息是否包括指定名字的 HTTP 头
String encodeRedirectURL(String)	在 sendRedirect 方法内封装特定的 URL，如果封装没有必要，返回无变化的 URL
String encodeURL(String)	使用 Session ID 编号来封装 URL，如果封装不必要，返回无变化的 URL
void sendError(int[,String])	用特定的状态码和默认提示信息返回客户端一个错误的响应信息；如果存在第二个参数，则用定义的信息取代默认的提示信息
void sendRedirect(String)	临时给客户端发送一个跳转到指定的 URL 的响应
void setDateHeader(String,long)	增加一个名-日期值对应的 HTTP 头
void setHeader(String,String)	增加一个名-值对应的 HTTP 头
void setContentType(String)	设置 MIME 类型
void setIntHeader(String,String)	增加一个名-整数值对应的 HTTP 头
void setStatus(int)	设置响应的状态代码

9.4.2 response 对象的应用实例

response 对象作为服务器返回 HTTP 响应的代表，通常与 out 对象和 request 对象结合可以实现输入输出信息的控制。request 对象获取客户请求信息（输入），out 对象和 response 对象结合实现响应客户（输出）。与 out 对象直接访问输出流的不同，response 对象是向客户端发送信息，根据不同的处理对客户端进行动态响应。在本节中，将对 response 对象处理 HTTP 响应头、response 对象使用 Cookies、response 重定向等方面展开实例介绍。

1. response 设置 HTTP 头信息与添加 Cookie

(1) 设置 HTTP 头信息。HTTP 头信息有两种不同情况。

一种是包含在客户端向服务器端提交 HTTP 请求。HTTP 请求头信息包括了客户端环境与请求实体的有用信息，具体包括 Accept（接受 Internet 媒体类型）、Accept-Language（可接受的语言）、Connection（联网状态）、Host（主机）、User-Agent（用户代理）、Content-Length（内容长度）、Content-Type（内容类型）、Accept-Encoding（接受的编码）等 HTTP 头以及对应的实体信息。

另外一种是在服务器向客户端发出响应产生的 HTTP 头信息。HTTP 响应头信息包括了 HTTP 响应头与响应实体的相关信息。常见的 HTTP 响应头有 Content-Type（内容类型）、Date（发送时间）、Etag（实体标记）、Expires（过期时间）、Host（主机）、Location（移动位置）、Refresh（刷新）、Server（服务器）和 Warning（响应补充信息）等。

response 对象有两个方法可以动态设置 HTTP 头信息。它们是 setHeader(String head,String value)和 addHeader(String head,String value)。这两个方法可以动态添加 HTTP 响应头和对应的值。值得注意的是，如果响应头已经存在，则后添加的响应头可以覆盖原来的内容。

(2) 添加 Cookie。Cookie 是服务器保存在客户端中的一小段数据信息，为服务器处理用户请求或追踪用户提供方便。JSP 页面通过调用 response 对象的 addCookie(Cookie)方法增加 Cookie。使用 Cookie 有一个前提，就是客户端浏览器允许使用 Cookie 并对此作出相应的设置。当前对使用 Cookie 存在较大的争议，一般不赞成使用 Cookie。由于 Cookie 仍有一定的应用空间，有了解的必要。在例 9-3 中，从一个简单实例来对以上内容做一个总结。

【例 9-3】 这是一个利用 response 对象设置 HTTP 文件头以及添加 Cookie 的应用实例。具体内容见程序清单 9-4，运行结果如图 9-4 所示。

程序清单 9-4：

```
<!--    JSP9-4.jsp    -->
<%@page contentType="text/html; charset=gb2312" language="java"  %>
<%@page import="java.io.*,javax.servlet.*,javax.servlet.http.*,java.util.*" %>
<html>
<head>
<title>response 对象的示例</title>
```

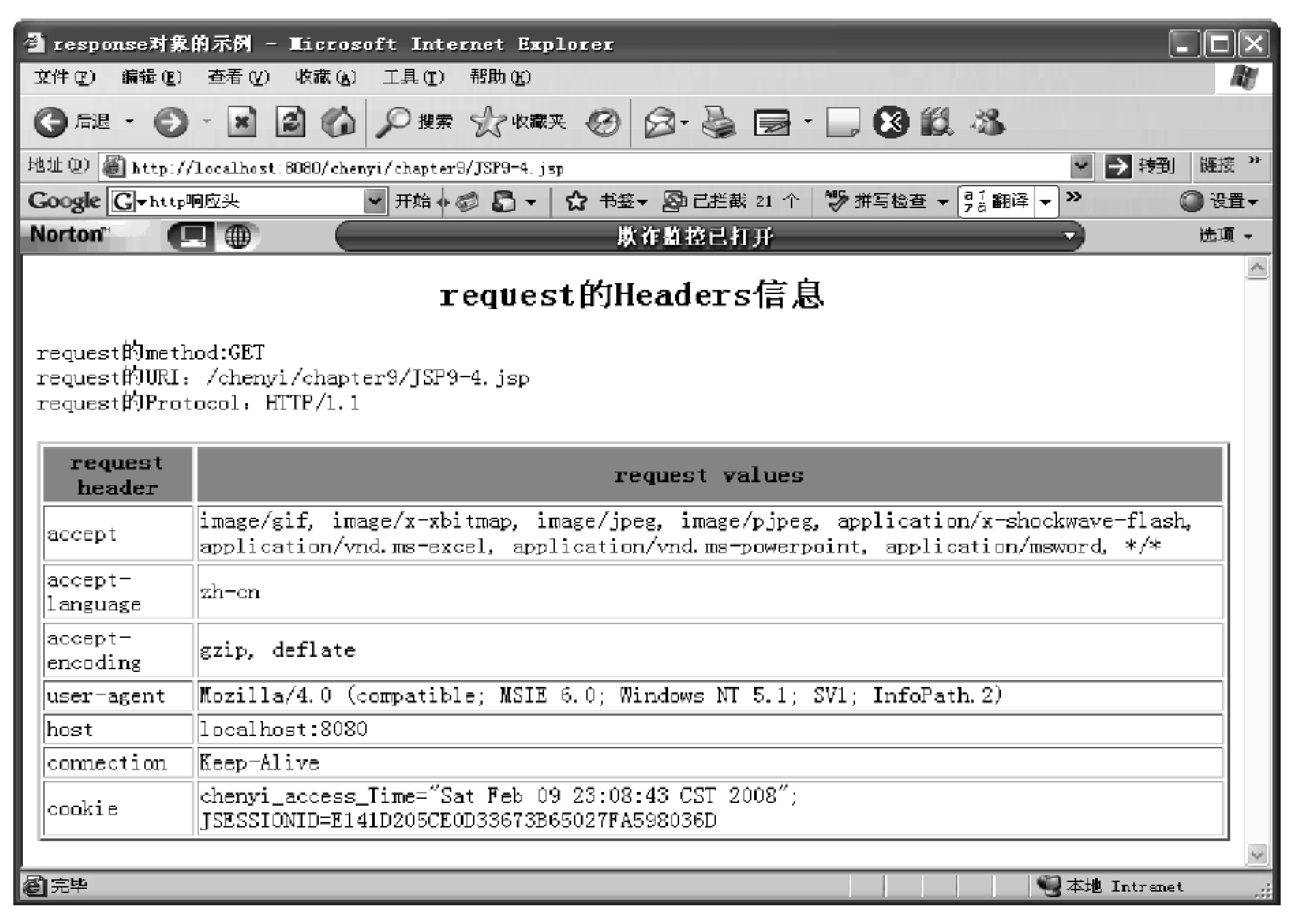
```
</head>
<body>
<%
response.setHeader("Refresh","5");          //设置 HTTP 响应头,每隔 5 秒刷新页面
Cookie cookie=null;                         //定义一个 cookie
if(cookie==null){
cookie=new Cookie("chenyi_access_Time",new Date().toString());
                                            //创建 cookie
response.addCookie(cookie);                 //增加 cookie
}
else{
  cookie.setValue(new Date().toString());   //设置 cookie
}
%>
<p><h2 align="center">request 的 Headers 信息</h2></p>
<p align="left">
request 的 method: <%=request.getMethod()%><br>
request 的 URI: <%=request.getRequestURI()%><br>
request 的 Protocol: <%=request.getProtocol()%><br>
</p>
<table border="2">
<th bgcolor="#FF6600">request header</th>
<th bgcolor="#FF6600">request values</th>
<%
  Enumeration headerNames=request.getHeaderNames();
  while(headerNames.hasMoreElements()){
      String headName=(String)headerNames.nextElement();
       out.println("<tr><td>"+headName);
       out.println("<td>"+request.getHeader(headName));
  }
%>
</table>
</body>
</html>
```

2. response 对象的重定向

response 对象通过 sendRedirect(String)实现重定向的作用。这意味着,从一个 JSP 页面跳转到由参数指定的 URL 指定的另一个 JSP 页面。从实现的效果上看,response 对象的重定向和<jsp:forward>的类似。但是两者有着明显的区别。

(1) response 对象是为客户端实现的跳转,首先将文件的所有内容完成。然后实现跳转,在实现跳转时,浏览器上的地址栏会发生变化。不传递参数。

(2) <jsp:forward>是为服务器端实现的跳转,发生跳转时,立即跳转到目标位置。在<jsp:forward>后面的内容将不会执行。另外,跳转发生时,浏览器的地址不会发生内容变化。但是可以传递信息。



(a)



(b)

图 9-4　利用 response 对象设置 HTTP 文件头以及添加 Cookie 的运行结果

【例 9-4】 response 对象的重定向应用实例。具体代码见程序清单 9-5。

程序清单 9-5：

```
<!--    JSP9-5.jsp    -->
<%@  page contentType="text/html; charset=gb2312" language="java"%>
```

```
<html>
<head>
<title>response重定向</title>
</head>

<body>
<%
   String name=request.getParameter("username");
   if(name==null)
     response.sendRedirect("login.jsp");    //无输入跳转到 login.jsp
   else if(name.equals("chenyi"))
     response.sendRedirect("welcome.jsp");  //验证成功跳转到 welcome.jsp
   else
     response.sendRedirect("error.jsp");    //验证失败跳转到 error.jsp
%>
</body>
</html>
```

9.5 session 对象

9.5.1 session 对象的概述

客户通过 HTTP 协议向 Web 服务器请求服务,而 Web 服务器对客户的请求发出响应。但是,HTTP 协议是一个无状态协议,当请求-响应完成时,连接就关闭了。对于连接相关信息服务器并没有保留。假设客户请求是一个需要登录的 Web 服务器,如果不间断地请求该服务器的服务,那么需要不停地登录。这无疑是影响客户的访问。服务器该如何跟踪用户,用以区分不同的用户成为一个必须解决的问题。实现跟踪用户有常见的 4 种方式:利用隐藏表单实现数据跟踪、重写包含参数的 URL、使用持续的 Cookie 和 session 会话机制。

利用隐藏表单和重写包含参数的 URL,将信息直接发给服务器,会带来大量的数据的交互,造成安全问题。而 Cookie 方式,即客户访问服务器后,服务器对于连接客户的相关信息写入到客户端的计算机中。但是,这存在一个问题,使用 Cookie 需要客户对客户端的浏览器进行 Cookie 启动的设置。在一定程度上限制了 Cookie 的应用。

JSP 支持 session 会话机制。session 会话机制实际上是客户访问 Web 网站,服务器会为每一个登录使用它的客户创建一个 session 会话对象。在这个 session 对象中记录了客户的相关信息。根据 session 对象记录的信息,服务器可以实现对客户的跟踪。当前客户退出服务器的服务时,对于该客户的 session 对象就会注销。而实现客户与服务器交互的这样一个过程就称为会话(session)。

JSP 的 session 对象就是在一次会话过程中创建的会话对象。从本质上说,session 对象是 Servlet API 的 javax.servlet.http.HttpSession 接口的对象实例。在一个 session 开始时,服务器端的 Servlet 容器就会创建一个 HttpSession 对象,即 session 对象,保存客户

状态信息。不同的客户，Servlet 容器会创建不同的 HttpSession 对象。为此，不同的 HttpSession 对象用不同的 session ID 进行区分。为了实现客户与服务器的 session 交互，session 对象中定义了常见的方法，这些方法见表 9-5。

表 9-5　session 对象的常见方法

方　　法	说　　明
long getCreationTime()	返回 session 对象创建的时间
String getId()	返回 session 对象的 session ID 编号
long getLastAccessedTime()	返回客户提交请求的最后时间
Object getValue(String)	返回 session 的应用层指定名称的值
String[] getValueNames()	返回 session 的应用层数据的名称
void invalidate()	使 session 对象无效并释放资源
boolean isNew()	判断是否是新创建的 session
void putValue(String,Object)	赋值对象数据给 session 的应用层指定的名称
void removeValue(String)	删除 session 的应用层指定名称的值
void setMaxInactiveInterval(int)	设置 session 处于不活动状态的最大时间间隔
int getMaxInactiveInterval()	返回 session 处于不活动状态的最大时间间隔
Enumeration getAttributeNames()	返回所有属性的名字
void setAttribute(String,Object)	设置指定属性的值
Object getAttribute(String name)	返回指定名字的属性，如果该属性不存在，将会返回 null
void removeAttribute(String name)	"删除指定的属性(包含属性名、属性值)。如果在有效时间内，用户做出了新的请求，那么服务器就会将其看作一个新的用户，此时，服务器将创建一个新的 session，旧的 session 信息将会丢失

9.5.2　session 对象的应用实例

session 对象表示特定会话 session 的用户的数据，提供了客户与服务器之间交互的联系。客户第一次访问支持 session 的 JSP 网页，服务器会创建一个 session 对象记录客户的信息。当客户访问同一网站的不同网页时，仍处于同一个 session 中。但是，当下列 3 种情况发生时，session 会注销：

(1) 客户关闭浏览器；

(2) 超过 session 生存时间；

(3) 以及在服务器端调用 invalidte()方法，强制使 session 无效。通常用这种方式实现用户注销。

在本节中，将通过例 9-5 的用户身份验证访问来了解具体的 session 的应用，以及 session 会话机制。

【例 9-5】　设计一个用户身份验证的应用。作用是用户输入用户名和密码信息登录网

站,验证登录信息,如果登录信息正确,则显示欢迎界面,否则返回登录界面。该应用由 4 个网页构成,如图 9-5 所示,运行结果如图 9-6～图 9-8 所示。

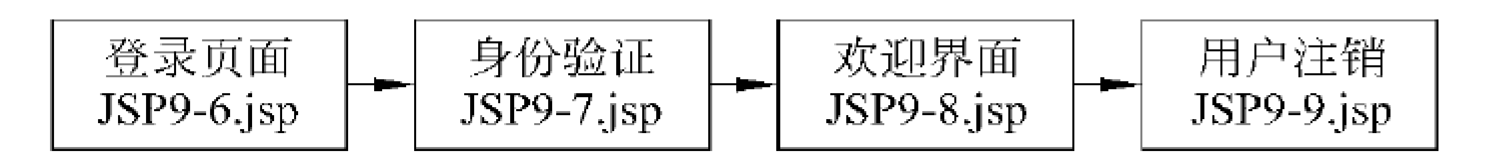


图 9-5　程序关系



图 9-6　登录界面

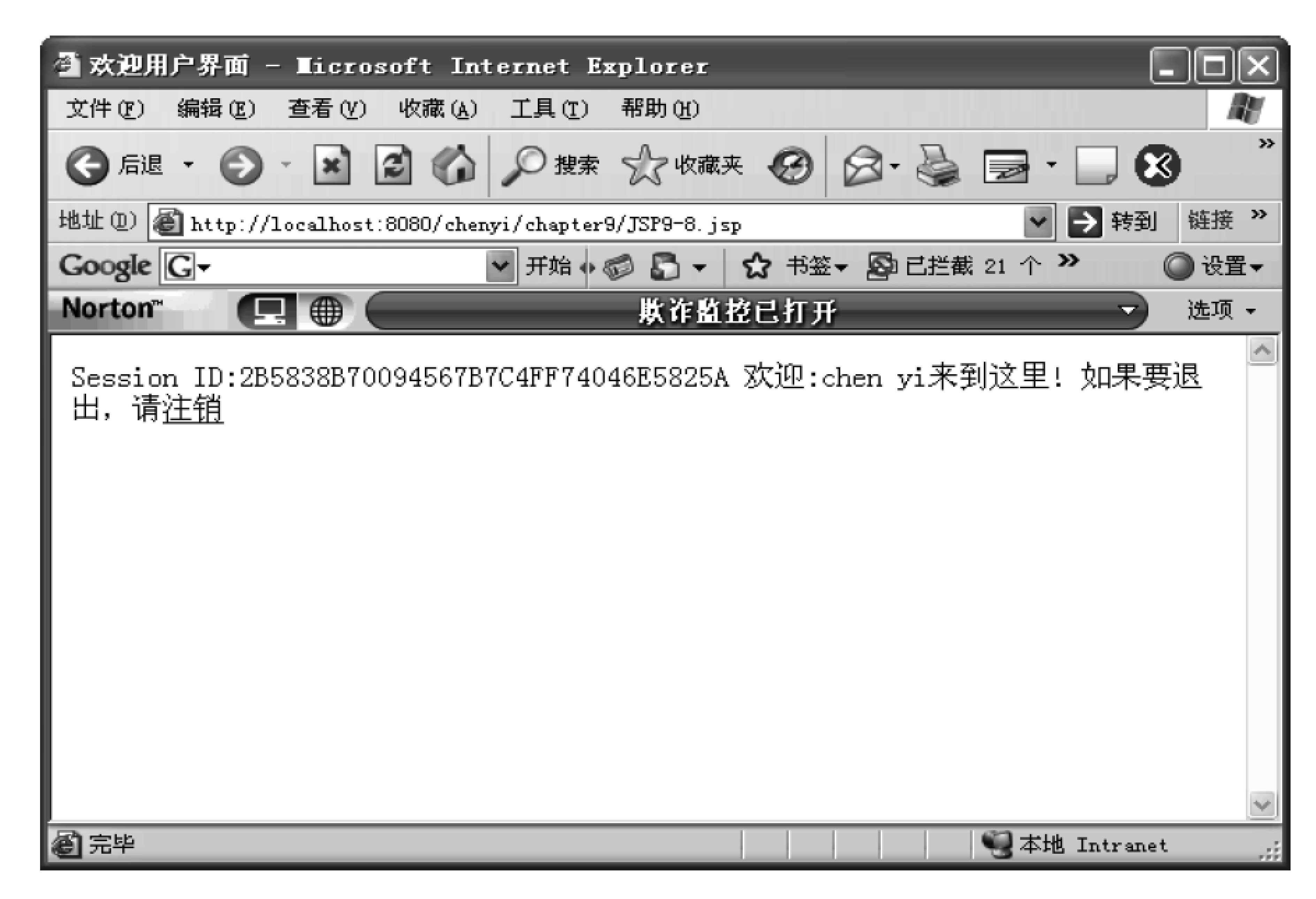


图 9-7　欢迎界面



图 9-8 注销界面

程序清单 9-6:

```
<!--    JSP9-6.jsp    -->
<%@ page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<title>用户登录界面</title>
</head>

<body>
<%
  if(!session.isNew()){
      String name=(String)session.getAttribute("username");
      if(name==null) name="";
  }
  out.println("Session ID: "+session.getId());    //输出会话编号
%>
<table border="1" align="center">
  <caption>用户登录:</caption>
<form action="JSP9-7.jsp" method="post">
      <tr>
        <td>用户名</td>
        <td>
          <input type="text" name="username"/>
        </td>
```

```
    </tr>
    <tr>
      <td>密码</td>
      <td><input type="password" name="password"/></td>
    </tr>
    <tr>
      <td><input type="submit" value="提交"/>
    </tr>
</form>
</table>
</body>
</html>
```

程序清单 9-7：

```
<!--    JSP9-7.jsp    -->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<title>用户身份验证</title>
</head>
<body>
  <%
   String username=request.getParameter("username");
   String password=request.getParameter("password");
   if(username==null) username="";
   if(password==null) password="";
   if(username.equals("chen yi")&&password.equals("123456")){ //验证用户信息
    session.setAttribute("username","chen yi");
    response.sendRedirect("JSP9-8.jsp");     //进入欢迎页面
   }
   else
    response.sendRedirect("JSP9-6.jsp");     //进入登录页面
   %>
</body>
</html>
```

程序清单 9-8：

```
<!--    JSP9-8.jsp    -->
<%@  page contentType="text/html; charset=gb2312" language="java" errorPage="" %>
<html>
<head>
<title>欢迎用户界面</title>
</head>
```

```
<body>
  <%
  String username= (String)session.getAttribute("username");
  out.println("Session ID: "+ session.getId());  //输出会话编号
  if(username!= null){
     out.println("欢迎: "+ username+ "来到这里!");
  }
  out.println("如果要退出,请<a href= 'JSP9-9.jsp'>注销</a>");
%>
  </body>
</html>
```

程序清单 9-9:

```
<!--    JSP9-9.jsp    -->
<%@  page contentType= "text/html; charset= gb2312" language= "java" %>
<html>
<head>
<title>欢迎用户界面</title>
</head>

<body>
  <%
  String username= (String)session.getAttribute("username");
  out.println("Session ID: "+ session.getId());  //输出会话编号
  session.invalidate();  //注销会话
  if(username!= null){
    out.println(username+ "再见!欢迎再来!");
}
%>
</body>
</html>
```

从上述的应用实例中可以看出,只要浏览器没有关闭或没有执行注销,所有网页的 Session ID 都是一致的。表示在这些会话过程中,服务器端的 Servlet 容器为客户创建的 session 对象发生作用。但是,请注意在实际身份验证中,需要结合数据库查询来实现。另外,在执行注销操作中,考虑用重定向跳转到网站的主页面会更加符合实际情况。

9.6 application 对象

application 对象是服务器的 Servlet 容器为多个应用程序保存信息。不同于 session 对象为一个客户端共享。application 对象的范围更大,在同一个服务器中的多个应用程序共享一个 application 对象。application 对象主要用于多个 Web 应用或者多个用户之间共享

数据。但是,当服务器关闭或重启动时,原有的 application 对象会被注销。

application 对象是实现 Servlet API 的 javax.servlet.ServletContext 接口的实例对象。application 的主要方法见表 9-6。

表 9-6　application 对象的常见方法

方　法	说　明
void setAttribute(String,Object)	设置属性
Object getAttribute(String)	返回指定属性的值
void removeAttribute(String)	删除属性
Enumeration getAttributeNames()	返回所有属性名
String getServerInfo()	返回网络服务 Servlet 名字和版本

【例 9-6】 用 application 对象实现一个计数器。具体代码见程序清单 9-10,运行结果如图 9-9 所示。



图 9-9　application 对象实现计数器的运行结果

程序清单 9-10:

```
<!--    JSP9-10.jsp    -->
<%@  page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<title>application对象的应用</title>
</head>

<body>
<p><h2>计数器</h2></p>
<%
  int count=1;
  try{
  count=Integer.parseInt(application.getAttribute("counter").toString());
```

```
    }catch(Exception e){
    out.println("<p align=center>计数器没有发生作用!</p>");
    }
    out.println("<p align=center>自从服务器启动后,此页面已经访问了"+count+"次</p>");
    count++;
    application.setAttribute("counter",count);
%>
</body>
</html>
```

9.7 config 对象

9.7.1 config 对象的概述

例 9.6 的 application 对象实现计数器存在一个问题,就是如果服务器关闭或重新启动,原有的 application 对象会注销。反映在计数器应用中就可以发现每一次启动服务器,计数器就要重新从 1 开始计算。这与客户访问网站的实际情况存在差距。如果重新启动服务器时,计数器能从服务器的配置中获取最后访问的次数,然后在这个基础上增加访问次数,无疑是一种符合实际情况的处理。

JSP 定义了 config 对象。config 对象可以获取服务器 Servlet 的相关配置。Tomcat 服务器的 Servlet 配置定义在对应 Web 应用的 WEB-INF 目录下 web.xml 文件中。简单地说,config 对象是在一个 servlet 初始化时,Servlet 引擎向它传递信息,具体包括 servlet 初始化时用的参数以及服务器的有关信息。还可以利用 config 对象实现日志记录,以及重定向控制等功能。config 对象是实现 javax.servlet.ServletConfig 接口的实例对象,它的常见方法见表 9-7。

表 9-7 config 对象的常见方法

方　　法	说　　明
String getInitParameter(String)	返回 servlet 的指定名称的初始化参数的值
Enumeration getInitParameterNames()	返回 servlet 的所有初始化参数的名字
ServletContext getServletContext()	返回 servlet 的上下文
String getServletName()	返回 servlet 对象实例的名称

9.7.2 config 对象的应用实例

config 对象可以获取 Servlet 的相关配置。用户可以根据实际需要在对应 Web 应用的 web.xml 文件中进行 Servlet 部署,为 config 对象获取 Servlet 相关信息创造条件。先从例 9-7 的简单实例来了解 config 对象以及对应的 Servlet 部署。

【例 9-7】 一个 config 对象获取所有 Servlet 的初始值的简单应用。部分 Servlet 的部署见当前的 Web 应用对应的 web.xml 文件,具体的 JSP 代码见程序清单 9-11,运行结果如

图 9-10 所示。

```
web.xml:
⋮
<servlet>
    <servlet-name>demo</servlet-name>                    <!--servlet名-->
    <jsp-file>/chapter9/JSP9-11.jsp</jsp-file>           <!--对应jsp文件-->
    <init-param>                                         <!--初始参数-->
            <param-name>name</param-name>
            <param-value>Chen Yi</param-value>
</init-param>
<init-param>
            <param-name>email</param-name>
            <param-value>XXXX@ yahoo.com</param-value>
     </init-param>
  </servlet>

  <servlet-mapping>                                      <!--servlet映射-->
     <servlet-name>demo</servlet-name>
     <url-pattern>/demo</url-pattern>                    <!--servlet对应url-->
  </servlet-mapping>
⋮
```

程序清单 9-11：

```
<!--    JSP9-11.jsp    -->
<%@ page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<title>JSP的config对象</title>
</head>

<body>
<h2 align="center">config的初始参数表</h2>
<table align="center" border="2">
<tr>
  <th bgcolor="# FF6600">参数名</th>
  <th bgcolor="# FF6600">值</th>
</tr>
<%
  Enumeration conenum=config.getInitParameterNames();
  while(conenum.hasMoreElements()){
  String str=(String)conenum.nextElement();
  out.println("<tr><td>"+str+"</td>");
  out.println("<td>"+config.getInitParameter(str)+"</td></tr>");
```

```
    }
%>
</table>
</body>
</html>
```



图 9-10　config 对象简单应用的运行结果

application 对象实现的计数器不符合实际情况。如果在 Web 应用的配置文件中如 web.xml 设置一个关于计数器的量，就可以用 config 对象获取该量的值。从而达到重新记数的功能，具体内容见例 9-8。

【例 9-8】 结合 config 对象和 application 对象实现计数器，部分 Servlet 的部署见当前的 Web 应用对应的 web.xml 文件，具体实现代码见程序清单 9-12，运行结果如图 9-11 所示。

```
web.xml:
 ⋮
  <servlet>
          <servlet-name>config_counter</servlet-name>
           <jsp-file>/chapter9/JSP9-12.jsp</jsp-file>
           <init-param>
              <param-name>counter</param-name>   <!--设置 counter-->
              <param-value>1000</param-value>    <!--从 1000 计数-->
           </init-param>
  </servlet>
  <servlet-mapping>
      <servlet-name>config_counter</servlet-name>
              <url-pattern>/config_counter</url-pattern>
  </servlet-mapping>
 ⋮
```

程序清单 9-12：

```
<!--   JSP9-12.jsp   -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>JSP的 config对象</title>
</head>
<body>
<%
  int servlet_counter=0;
  int count=0;
  try{
        servlet_counter=Integer.parseInt(config.getInitParameter("counter"));
                                                            //获取 counter的值
              }catch(Exception e){
              }
  try{
        count=Integer.parseInt(application.getAttribute("config_counter").toString());
    //获取 application对象的 config_counter属性的值
              }catch(Exception e){
              }
  if(count<servlet_counter) count=servlet_counter;
  out.println("此页面已经访问了"+count+"次");
    count++;
  application.setAttribute("config_counter",new Integer(count));
%>
</body>
</html>
```



图 9-11 实现的计数器的运行结果

如同程序清单 9-12 展示的，config 对象获取 servlet 中的 counter 参数对应的值，并将该值给予变量 servlet_counter。然后与服务器产生的 application 对象记录的计数器

属性的值进行比较，将较大的值保存到 application 的记录计数器属性的值 config_counter 中。通过 config 对象，即使服务器重新启动，也可以保证计数器可以从 servlet 配置的 1000 开始。如果修改 servlet 配置的 web. xml 为实际的运行次数，会与实际情况相近。

9.8 exception 对象

9.8.1 exception 对象的概述

exception 对象是 java. lang. Throwable 类的一个实例，表示运行时的异常。exception 对象用来处理 JSP 文件在执行时发生的错误和异常。作为 java. lang. Throwable 的实例对象，exception 的常见方法见表 9-8。

表 9-8 exception 对象的常见方法

方　法	说　明
String getMessage()	返回错误信息
void printStackTrace()	以标准错误的形式输出一个错误和错误的堆栈
String toString()	以字符串的形式返回一个对异常的描述

exception 对象可以配合 page 指令一起使用，page 指令指定某一个页面为错误处理页面。把所有的错误都集中那个页面进行处理，加强整个系统的健壮性，以及使得程序的流程更加简单明晰。当 JSP 页面发生运行错误时，Servlet 容器会自动调用 page 指令指定的错误处理页面。在 9.8.2 小节中将对 exception 对象的具体应用做详细介绍。

9.8.2 exception 对象的应用实例

JSP 页面中错误调用与错误处理是一个重要的应用。通过 exception 对象可以追踪运行过程中存在的问题，从而做出相应的处理。在这样的一个处理流程中，体现出错页面与错误处理页面之间的关系。出错页面出现错误，会调用错误处理的 JSP 页面。而错误处理的页面可以通过 exception 对象来获得错误信息以及执行相关的处理动作。

要实现这样的错误处理的流程：首先，在出错的 JSP 页面中，需要 page 指令进行设置属性 errorPage 来指定错误处理的页面；而在错误处理页面中，要用 page 指令设置 isErrorPage 属性的值为“true”，通过这样的设置，才允许 exception 对象的应用。在下列的例 9-9 中对此做了一个说明。

【例 9-9】 设计一个登录界面，如果用户名或密码名为空，转向错误处理页面进行错误处理。本应用由三个程序构成：(1)登录界面程序，见程序清单 9-13；(2)验证用户名或密码名是否为空，见程序清单 9-14；(3)错误处理的页面，见程序清单 9-15，运行结果如图 9-12 和图 9-13 所示。

程序清单 9-13：

```
<!--    JSP9-13.jsp    -->
<%@ page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<title>用户登录界面</title>
</head>
<body>
<table border="1">
  <caption>
      用户登录：
  </caption>
<form action="JSP9-14.jsp" method="post">
    <tr>
      <td>用户名</td>
      <td><input type="text" name="username"/></td>
    </tr>
    <tr>
      <td>密码</td>
      <td><input type="password" name="password"/></td>
    </tr>
    <tr>
      <td><input type="submit" value="提交"/>
    </tr>
</form>
</table>
</body>
</html>
```



图 9-12　登录界面



图 9-13 错误处理页面运行结果

程序清单 9-14：

```
<!--    JSP9-14.jsp    -->
<%@ page contentType="text/html; charset=gb2312" language="java"  errorPage="JSP9-15.jsp" %>
<html>
<head>
<title>出错页面</title>
</head>

<body>
<%
  String username=request.getParameter("username");
  String password=request.getParameter("password");
  if(username==null) username="";
  if(password==null) password="";
  String errorMessage="";
  if(username.equals(""))
    errorMessage+="警告：用户名不能为空!";
  if(password.equals(""))
    errorMessage+="\n警告：密码不能为空!";
  if(!errorMessage.equals(""))
    throw new Exception(errorMessage);
%>
</body>
</html>
```

程序清单 9-15：

```
<!--    JSP9-15.jsp    -->
<%@ page  language="java" isErrorPage="true"%>
<html>
<head>
<title>错误处理页面</title>
</head>
<body>
```

```
<%
  out.println(exception.getMessage()); //错误处理
%>
</body>
</html>
```

有一个情况需要注意，运行上述程序，如果运行的是默认配置的 IE 5.0 以上浏览器，则并不会出现用户自定义的错误页面结果。只会在浏览器中出现“500 内部错误”。这是因为 IE 默认设置中使用了友好 HTTP 错误信息导致。如果需要使用用户自定义的错误信息，需要将 IE 浏览器的“工具 Internet 选项-高级-显示友好 HTTP 错误信息”选项信息取消，用户自定义的错误页面才可以使用。

9.9 page 对象

page 对象是 java.lang.Object 的对象实例，它是 JSP 的实现类的实例，即 JSP 的 Servlet 处理当前请求的对象实例。JSP 文件会由 Servlet 容器在第一次运行时编译成 Servlet 类，每次运行会创建一个该类的对象，这个对象可以用 page 表示。page 对象相当于是编译的 Servlet 中 this 引用的一个代名词。该对象很少使用。

9.10 pageContext 对象

pageContext 对象是 javax.servlet.jsp.pageContext 类的一个对象实例。该内置对象提供 JSP 页面上下文，表示 JSP 页面本身。它可以实现对 JSP 页面内所有的对象以及属性的管理和访问。pageContext 对象的常见方法见表 9-9。

表 9-9 pageContext 对象的常见方法

方　法	说　明
Object getAttribute()	返回与指定范围内名称有关的变量或 null
Object findAttribute(String)	用来按照页面请求、会话以及应用程序范围的顺序实现对某个已经命名属性的搜索
void setAttribute(String,Object)	用来设置默认页面的范围或指定范围之中的已命名对象
void removeAttribute()	用来删除默认页面范围或指定范围之中已命名的对象
Exception getException()	返回当前 exception 对象
ServletRequest getRequest()	返回当前的 request 对象
ServletResponse getResponse()	返回当前的 response 对象
ServletConfig getServletConfig()	返回当前页面的 servletConfig 对象，如 config 对象
ServletContext getServletContext()	返回当前页面的上下文 ServletContext 对象，如 application 对象
HttpSession getSession()	返回当前页面的 session 对象

【例 9-10】 一个 pageContext 简单应用实例。具体内容见程序清单 9-16，运行结果如

图 9-14 所示。

程序清单 9-16：

```
<!--    JSP9-16.jsp    -->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<title>pageContext 的应用</title>
</head>

<body>
<p align="center">contextPage 对象的应用示例</p>
<%
   pageContext.setAttribute("name",new String("陈铁"));
   pageContext.setAttribute("userId",new String("chan_yee"));
%>
<table align="center" border="2">
<tr><th bgcolor="# FF6600">参数</th>
    <th bgcolor="# FF6600">值</th></tr>
<tr>
<td>姓名</td>
<td><%=pageContext.getAttribute("name")%></td>
</tr>
<tr>
<td>用户号</td>
<td><%=pageContext.getAttribute("userId")%></td>
</tr>
</table>
</body>
</html>
```



图 9-14　pageContext 对象应用运行结果

小　　结

到目前为止，了解了 JSP 的 9 种内置对象。这 9 种内置对象的含义、作用都有不同。此外，9 种内置对象各有各的作用域。属于 page 范围的对象有 out、response、page、pageContext、config 以及 exception。这些对象只对应用它们的页面有效；属于 request 范围的对象有 request。对于 request 对象在一次用户请求中有效。属于 session 作用范围的对象有 session 以及属于 application 范围的对象有 application。这些内置对象不同结合可以实现输入输出的控制、文件流的控制、会话管理以及日志和错误处理、初始化参数的管理等多方面的应用。本章通过具体实例对这些应用做了初步的介绍。

习　题　9

1. 填空题

（1）网站追踪用户常见有________、________、________和________这 4 种方法。

（2）session 是________。

（3）HTTP 协议是一个________(无|有)连接的通信协议。

（4）response 对象是________接口的实例。

（5）调用 page 对象的 getSession()方法的结果是________。

2. 程序填空

（1）下列代码错误的是第________行。

```
第 1行：<%@  page  language="java"%>
第 2行：<html>
第 3行：<head>
第 4行：<title>错误处理页面</title>
第 5行：</head>
第 6行：<body>
第 7行：<%
第 8行：out.println(exception.getMessage());  //错误处理
第 9行：%>
第 10行：</body>
第 11行：</html>
```

（2）下列程序段可以实现计数器的功能，请将空白处补充完整。

```
<%@page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>计数器</title>
</head>
<body>
```

```
<p><h2>计数器</h2></p>
<%   int count=1;
    try{
    count=Integer.parseInt(______①______);
    }catch(Exception e){
    out.println("<p align=center>计数器没有发生作用!</p>");
    }
    out.println("<p align=center>自从服务器启动后,此页面已经访问了"+count+"次</p>");
    count++;
    ______②______
%>
</body>
</html>
```

3. 实验题

用内置对象设计一个简单的留言板。要求不使用数据库来实现。

第 10 章　JSP 的文件操作

10.1　File 类

在 Java 语言的程序设计中，IO 操作，特别是文件操作是非常重要的一部分，同样，文件操作在 JSP 中也占据着非常重要的地位。本章将介绍 JSP 中的文件操作，由于 JSP 能使用 Java 的 IO 系统，所以在 JSP 中对文件的操作和一般 Java 程序没有太大的区别。首先介绍 java. io. File 类，java. io. File 类专门提供一种抽象，用于以平台独立的方式处理大多数平台依赖的、复杂的文件和路径名问题。File 类包含许多获取文件属性的方法以及创建、删除和重命名文件的方法，但是 File 类没有包含读写文件内容的方法。

10.1.1　获取文件属性

存储在变量、数组和对象中的数据是暂时的，程序结束后它们就会丢失。为了能够永久地保存程序创建的数据，需要将它们存储到磁盘文件或光盘文件中。以后可以传送这些文件，也可以在其他程序中使用这些文件。在文件系统中，每个文件都存放在一个目录下，文件名全称是由目录路径和文件名组成的。例如 c：\jspbook\filelist. jsp 是文件 filelist. jsp 在 Windows 操作系统上的文件名全称，其中 c：\jspbook 称为该文件的目录路径。目录路径和文件名是依赖于平台的，例如，在 Linux 系统中 filelist. jsp 的文件名全称可能是/home/user/jspbook/filelist. jsp，其中/home/user/jspbook 是文件 filelist. jsp 的目录路径。

1. File 的创建

文件名是一个字符串，File 类是文件名及其目录路径的一个包装类。它的主要构造方法如下。

(1) File(String pathname) 通过将给定路径名字符串转换成抽象路径名来创建一个新 File 实例。

(2) File(String parent，String child) 根据 parent 路径名字符串和 child 路径名字符串创建一个新 File 实例。

(3) File(File parent， String child) 根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例。例如，在 Windows 中，语句“new File("c：\\jspbook\\filelist. jsp")”为 filelist. jsp 文件创建了一个 File 实例，注意，在 Windows 中，反斜杠是一个特殊的字符，应该使用转义字符“\\”表示实际的“\”。可以用 File 类的 isDirectory()方法来判断一个对象是否表示目录，用 isFile()来判断一个对象是否表示文件。

2. File 的主要方法

File 的主要方法见表 10-1。

表 10-1　File 的主要方法

方　　法	功　　能
boolean canRead()	测试应用程序是否能从指定的文件中进行读取
boolean canWrite()	测试应用程序是否能写当前文件
boolean delete()	删除当前对象指定的文件
boolean exists()	测试当前 File 是否存在
String getAbsolutePath()	返回由该对象表示的文件的绝对路径名
String getCanonicalPath()	返回当前 File 对象的路径名的规范格式
String getName()	返回表示当前对象的文件名
String getParent()	返回当前 File 对象路径名的父路径名，如果此名没有父路径则为 null
String getPath()	返回表示当前对象的路径名
boolean isAbsolute()	测试当前 File 对象表示的文件是否是一个绝对路径名
boolean isDirectory()	测试当前 File 对象表示的文件是否是一个路径
boolean isFile()	测试当前 File 对象表示的文件是否是一个普通文件
long lastModified()	返回当前 File 对象表示的文件最后修改的时间
long length()	返回当前 File 对象表示的文件长度
String[] list()	返回当前 File 对象指定的路径文件列表
File[] listFiles()	返回一个抽象路径名数组，这些路径名表示此抽象路径名所表示目录中的文件
boolean mkdir()	创建一个目录，它的路径名由当前 File 对象指定
boolean renameTo(File)	将当前 File 对象指定的文件更名为给定参数 File 指定的路径名
String toString()	返回当前对象的字符串表示

File 类有 4 个常量：pathSeparator、pathSeparatorChar、separator 和 separatorChar。它们是平台无关的路径和文件名分隔符。separatorChar 在 Windows 上是'\'，而在 Linux 上是'/'。separatorChar 是一个字符，separator 是一个表示 separatorChar 的字符串。类似地，pathSeparator 是表示 pathSeparatorChar 的字符串，pathSeparatorChar 在 Windows 上是';'，而在 UNIX 上是':'。

绝对路径名是依赖于系统的。在程序中，不要直接使用绝对目录名和文件名，因为它能在 Windows 中运行，而不能在其他系统上运行。为了能让程序在不同的平台上正常运行，使用下列字符串代替“c：\\book\\test.dat”。

```
new File(".").getCanonicalPath()+"book"+File.separator+"test.dat";
```

其中“.”表示当前目录。

程序清单 10-1 演示了一个使用 File 类显示文件或者文件夹属性的 JSP 程序。

程序清单 10-1：

```
<!--file.html-->
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>显示文件属性</title>
</head>
<body>
<form name="form1" method="post" action="file.jsp">
  <table width="400" border="0" cellspacing="0" cellpadding="0">
    <tr>
      <td>选择文件或者文件夹:</td>
      <td><input type="file" name="filename"></td>
    </tr>
    <tr>
      <td rowspan="2"><input type="submit" name="Submit" value="提交"></td>
    </tr>
  </table>
</form>
</body>
</html>
<!--file.jsp-->
<%@ page contentType="text/html; charset=gb2312"
language="java"%>
<%@ import="java.io.*,java.util.*"%>
<html>
<head>
<title>文件属性</title>
</head>
<body>
<%
String dir=request.getParameter("filename");
File f=new File(dir);
if(f.exists()){
%>
<%=f.getName()%>的属性如下:<br>
文件路径:<%=f.getCanonicalPath()%><br>
文件长度为:<%=f.length()%>bytes<br>
是文件吗?:<%=f.isFile()%><br>
是目录吗?:<%=f.isDirectory()%><br>
可读取?:<%=f.canRead()%><br>
可写入?:<%=f.canWrite()%><br>
文件的最后修改日期为:<%=new Date(f.lastModified())%><br>
<%}%>
</body>
</html>
```

程序清单 10-1 中 file.html 设计了一个表单，用文件输入框打开相应的文件，并提交到

file.jsp 文件，运行结果如图 10-1 所示。在 file.jsp 中，创建了一个 File 实例，并返回了它的相关属性，运行结果如图 10-2 所示。



图 10-1 file.html 选择文件表单

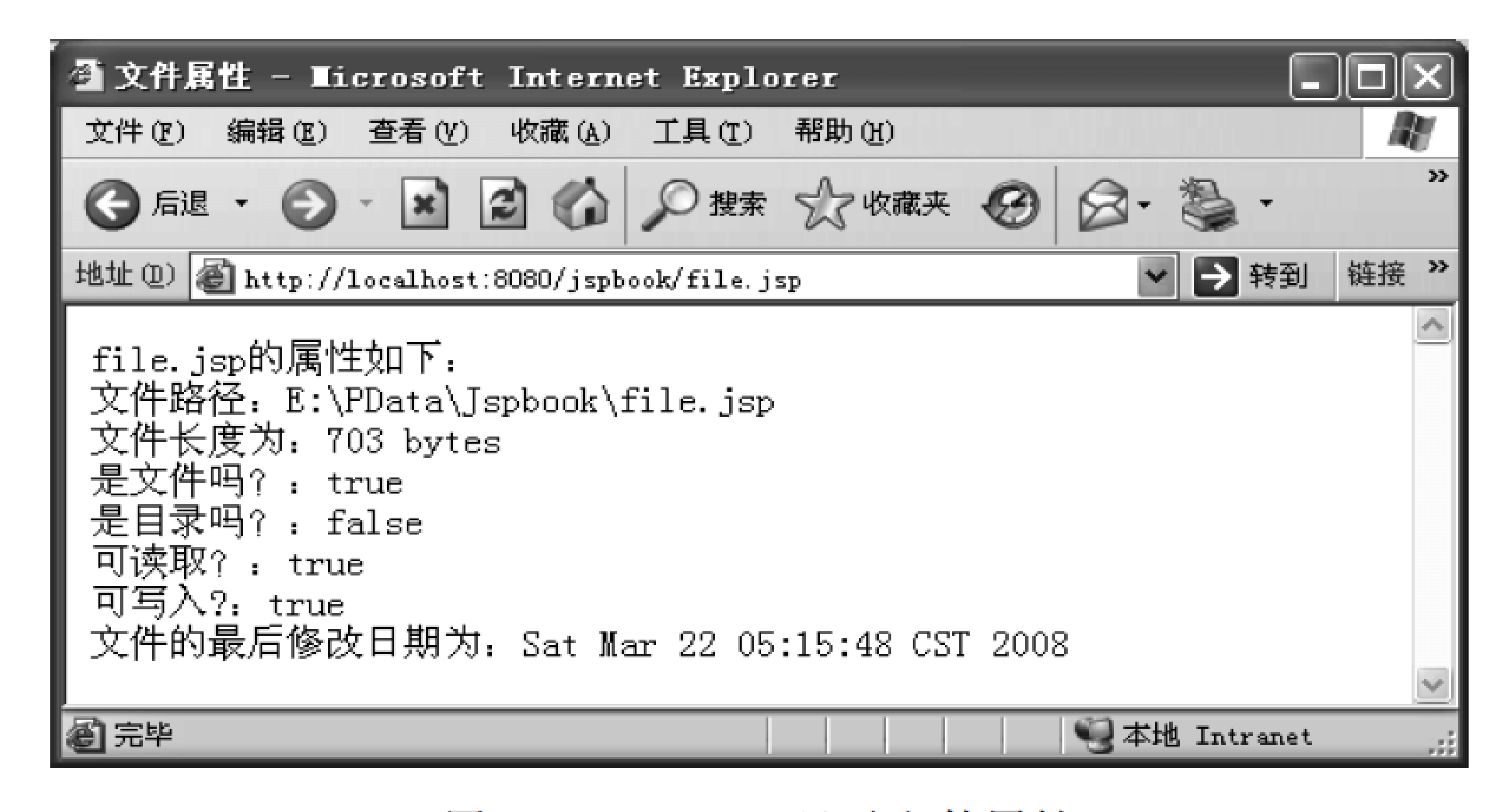


图 10-2 file.jsp 显示文件属性

10.1.2 创建目录

File 同样支持对目录的操作，isDirectory()方法可以用来判断是否是一个目录，makdir()方法，实现对文件目录的建立，建立成功后返回 ture，否则返回 false。

程序清单 10-2 演示了在服务器当前目录建立一个新的子目录的方法。

程序清单 10-2：

```
<!--dircreate.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.io.*"%>
<html>
<head>
<title>目录建立实例</title>
</head>
<body>
<%
String path=getServletContext().getRealPath(".");    //获得当前路径
File f=new File(path,File.separator+"newDir");       //包装将建立的目录路径 File类
```

```
    if(f!=null&&!f.exists()){                          //检查 Sub 目录是否存在
      if(f.mkdir()){
          out.println("服务器当前路径"+path+"中<br>");
        out.println("newDir 目录创建成功");
      }
    }
%>
</body>
</html>
```



图 10-3 创建目录

程序清单 10-2 中 dircreate.jsp 文件在当前目录中建立子目录“newDir”，输出结果如图 10-3 所示。程序中 getServletContext().getRealPath(".")方法获得当前的路径，也可以用于获得其他的路径，如 getServletContext().getRealPath("../qqq/www/")获得了"../qqq/www/"，当然"../qqq/www/"必须真实存在。程序中用

```
File f=new File(path,File.separator+"newDir");
```

建立需要创建目录的 File 实例，其中 File.separator 表示目录分隔符，在 Windows 中是“\”，这里也可以直接采用如下语句来创建 File 实例，

```
File f=new File(path, "\\newDir");
```

但是考虑到平台独立性，还是推荐使用平台无关的分隔符 separator。

10.1.3 删除文件和目录

File 中的 delete()方法用于删除此抽象路径名表示的文件或目录。如果此路径名表示一个目录，则此目录必须为空才能删除。下列语句表示删除对应的文件或者目录：

```
File fileNeedDelete=new File(文件或目录路径);
if(fileNeedDelete.exists()){
  fileNeedDelete.delete();
}
```

10.2 JSP 的输入流和输出流

File 对象封装了文件或路径属性，但是不包含从文件中读写数据的方法。为了进行文件读写操作，需要用适当的 Java I/O 类创建对象，这些对象包含从文件中读写数据的方法。Java 有许多用于各种目的的 I/O 类，可以把它们分为输入类和输出类。输入类包含读数据的方法，输出类包含写数据的方法。例如，要将文本写入名为 dat.txt 的文本文件中，可以使用 java.io.FileWriter 类按如下方式创建一个对象：

```
FileWriter out=new FileWriter("dat.txt");
```

然后可以使用 writer()方法向文件写入一个字符串。如下代码将“I like Jsp”写入 dat.txt 文件：

```
out.write("I like Jsp");
```

当写入操作完成后，应该关闭输出对象 out：

```
out.close();
```

同样的，如果要从一个文件里读入字符，则可以建立一个 java.io.FileReader 输入类对象，并使用 read()方法来读入字符：

```
FileReader in=new FileReader("dat.txt");
char c=in.read();
in.close();
```

上述代码中的 read()方法从文本文件里面读入一个字符，如果文件中是“I like Jsp”，则读入第一个字符'I'。

从上面的例子可以看到在 Java 中是使用建立 I/O 对象的方式进行输入和输出，输入的对象被称为输入流，输出的对象被称为输出流。

流是一个很形象的概念，当程序需要读取数据时，就会开启一个通向数据源的流，这个数据源可以是文件，内存，或是网络连接，这就是“输入流”，如图 10-4 所示。类似的，当程序需要写入数据的时候，就会开启一个通向目的地的流，这就是“输出流”，如图 10-5 所示，这时你就可以想象数据好像在这其中“流”动一样。

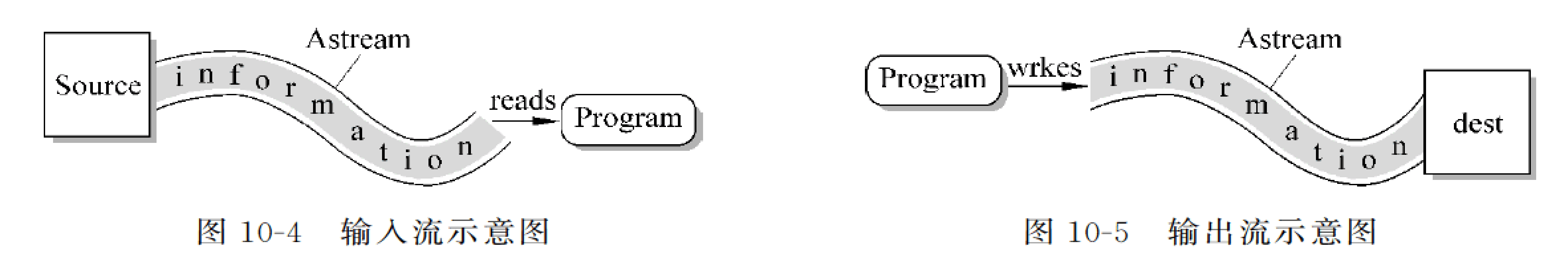


图 10-4 输入流示意图　　图 10-5 输出流示意图

Java I/O 中的流分为两种，一种是字节流，另一种是字符流，分别由 4 个抽象类来表示：InputStream、OutputStream、Reader 和 Writer。Java 中其他多种多样变化的流均是由它们继承出来的。

10.2.1 字节流

以字节为单位进行读写的称为字节流，在 Java 中，字节流分为“输入流”和“输出流”。可以从中读出一系列字节的对象称为“输入流”(InputStream)；而能向其中写入一系列字节的对象则称为“输出流”(OutputStream)。这两种对象分别是由 java.io 包中的抽象类 InputStream 和 OutputStream 来实现的。部分字节流结构如图 10-6 和图 10-7 所示。从图中可以看出。

(1) 字节流的两个顶层类是抽象类，分别是 InputStream 和 OutputStream。

(2) 每个抽象类都有子类来实现具体的功能，处理不同的输入和输出。InputStream 和 OutputStream 类是不能实例化的，实际上在程序中使用的就是它们的子类，如涉及文件输入和输出的是文件输入输出流 FileInputStream 和 FileOutputStream。

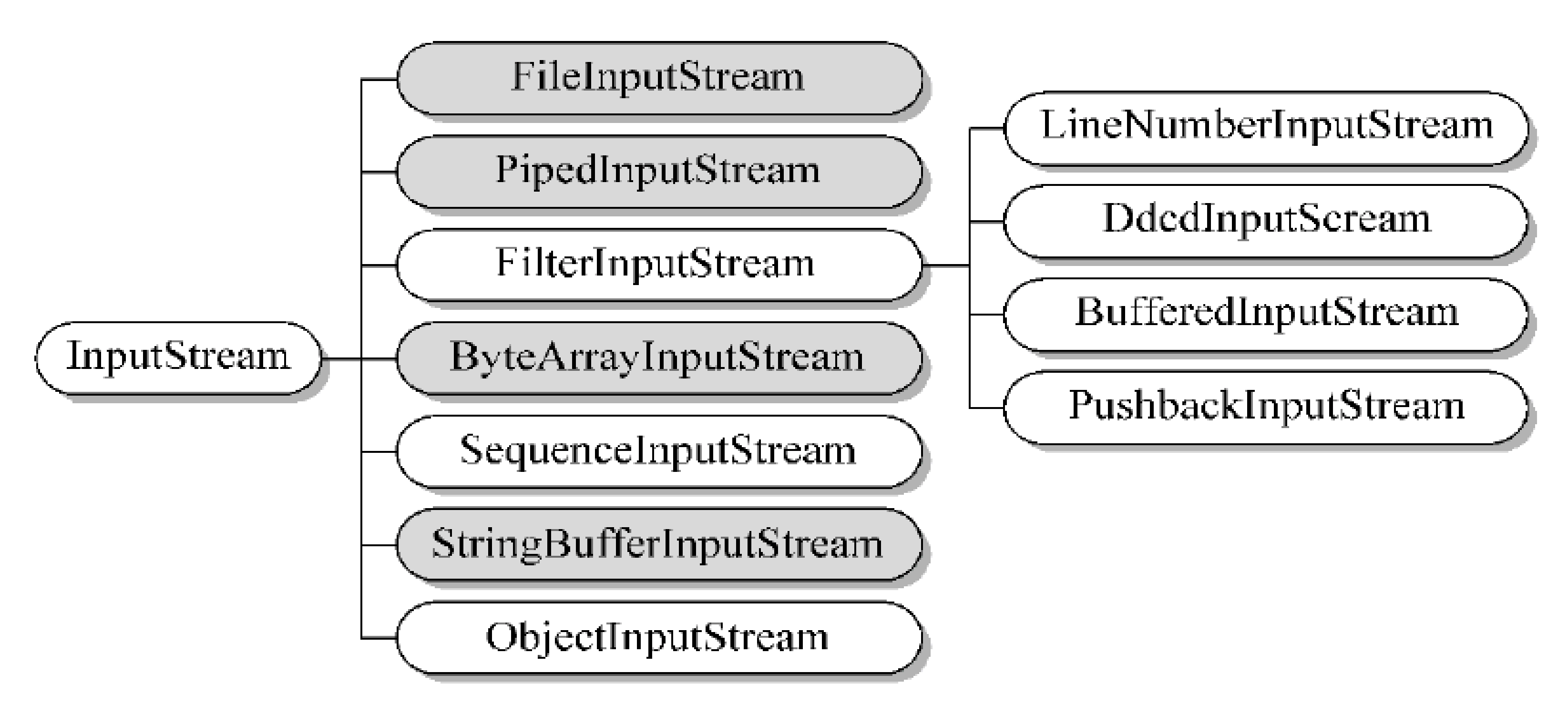


图 10-6　InputStream 类结构图

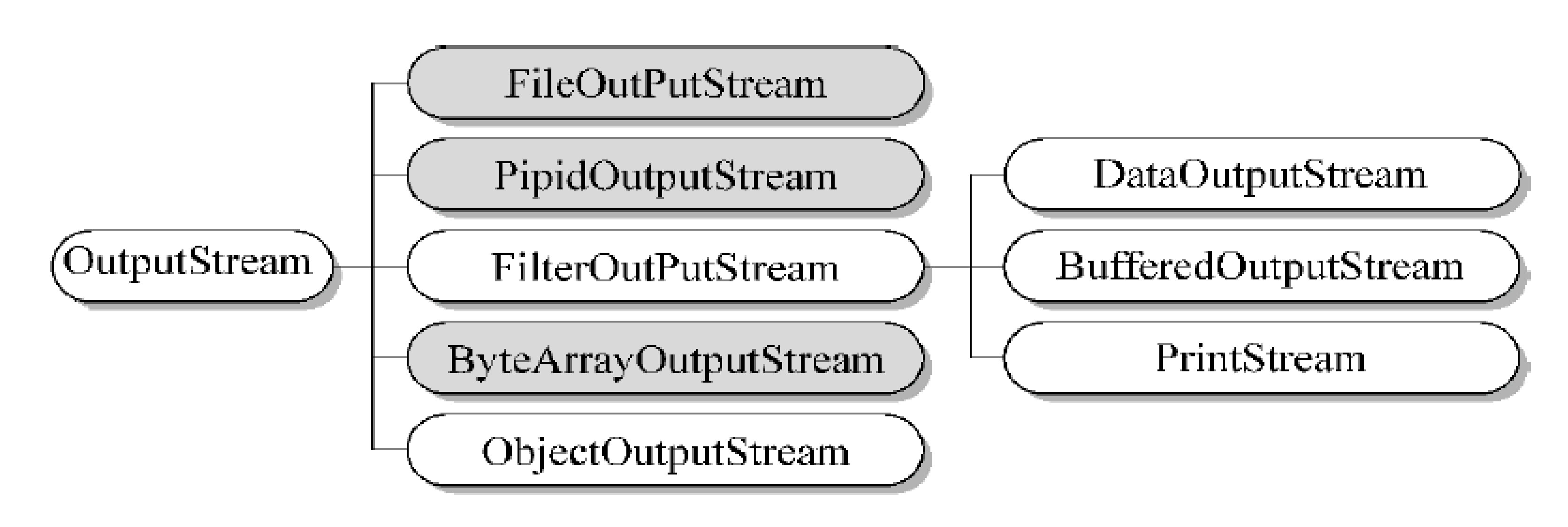


图 10-7　OutputStream 类结构图

表 10-2 是字节流的几个常用子类及功能。

表 10-2　字节流常用子类及功能

字 节 流 类	功能简单介绍
DataInputStream	包含了读取 Java 标准数据类型的输入流
DataOutputStream	包含了写入 Java 标准数据类型的输出流
ByteArrayInputStream	从字节数组读取的输入流
ByteArrayOutputStream	写入字节数组的输出流
FileInputStream	从文件读入的输入流
FileOutputStream	写入文件的输出流
PrintStream	包含最常见的 print()和 println()的输出流
PushbackInputStream	返回一个字节到输入流,主要用于编译器的实现
PipedInputStream	输入管道
PipedOutputStream	输出管道
SequenceInputStream	将 n 个输入流联合起来,一个接一个按一定顺序读取
BufferInputStream	缓冲输入流
BufferOutputStream	缓冲输出流
FilterInputStream	实现了 InputStream 接口的过滤器输入流
FilterOutputStream	实现了 OutputStream 接口的过滤器输出流

字节流抽象类 InputStream 和 OutputStream 的常用方法如表 10-3 所示。

表 10-3　InputStream 和 OutputStream 的常用方法

方　　法	功　　能
int read()	读取一个字节的数据，并返回读到的字节
int read(byte[] b)	将数据读入一个字节数组，同时返回读回的字节数
int read(byte[] b, int off, int len)	将数据读入一个字节数组。返回读回的实际字节数。其中，b：指定要把字节读入哪个数组；off：指定数组的偏移位置，第一个字节应放在哪个位置；len：读回的最大字节数
long skip(long n)	在输入流中跳过几个字节，它返回的实际跳过的字节数
int available()	返回在不加阻止的情况下，可用的字节数
void mark(int readlimit)	在此输入流中标记当前的位置
void reset()	将此流重新定位到对此输入流最后调用 mark 方法时的位置
long skip(long n)	跳过和放弃此输入流中的 n 个数据字节
void write(int b)	写入一个字节的数据
void write(byte[] b)	写入数组 b 内的所有字节
void write(byte[] b, int off, int len)	写入数组 b 内的所有字节，其中，b：指定要从哪个数组写入数据；off：指定数组 b 的一个偏移位置，从哪个位置的字节开始写入；len：要写入的字节数量
void flush()	清空输出流
void close()	关闭输入流或者输出流

关于基本输入流和输出流的方法还有两点需要说明。

(1) read()方法，如果没有数据可读(即 available()返回 0 时)则输入流将被阻塞，直到有新的数据可读时才继续执行；

(2) InputStream 和 OutputStream 中的方法若发生输入输出错误时抛出 IOException。读者应该在程序中注意捕捉和处理。

10.2.2　字符流

Java 中的字节流是用于处理字节的输入和输出的，包括读写二进制数据等方面的内容。而 Java 中的字符流则用于处理字符的输入和输出，采用的是 Unicode 编码，可以实现国际化。使用字符流的另外一个好处就是：字符流比字节流更有效率。

同字节流类似，字符流也是通过两个顶层的抽象类 Reader 和 Writer 的子类来实现对 Unicode 字符流的处理。如图 10-8 和图 10-9 所示。

从图中可以看出：

(1) 字符流的两个顶层类是抽象类，分别是 Reader 和 Writer。

(2) 每个抽象类都由子类来实现具体的功能，处理不同的设备的输入和输出。

表 10-4 是字符流的几个常用子类及功能。

Reader 和 Writer 的常用方法与表 10-3 中 InputStream 和 OutputStream 的常用方法

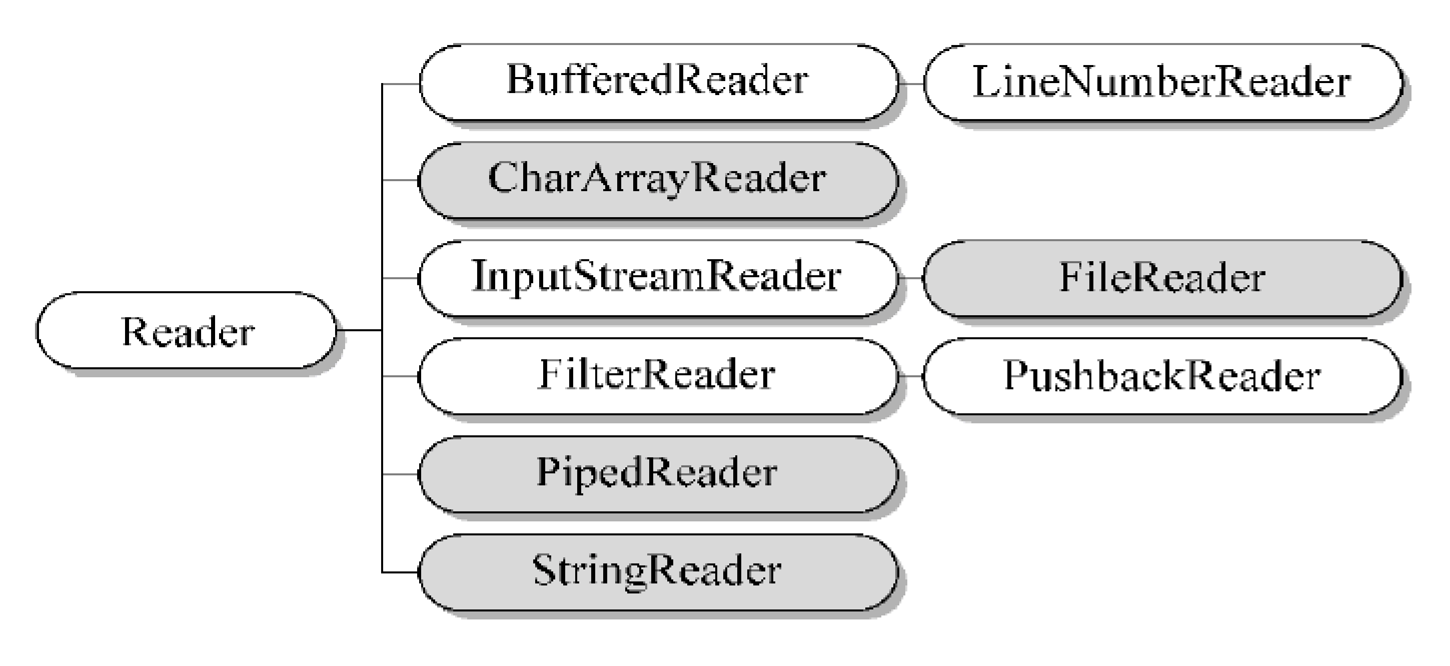


图 10-8　Reader 类结构图

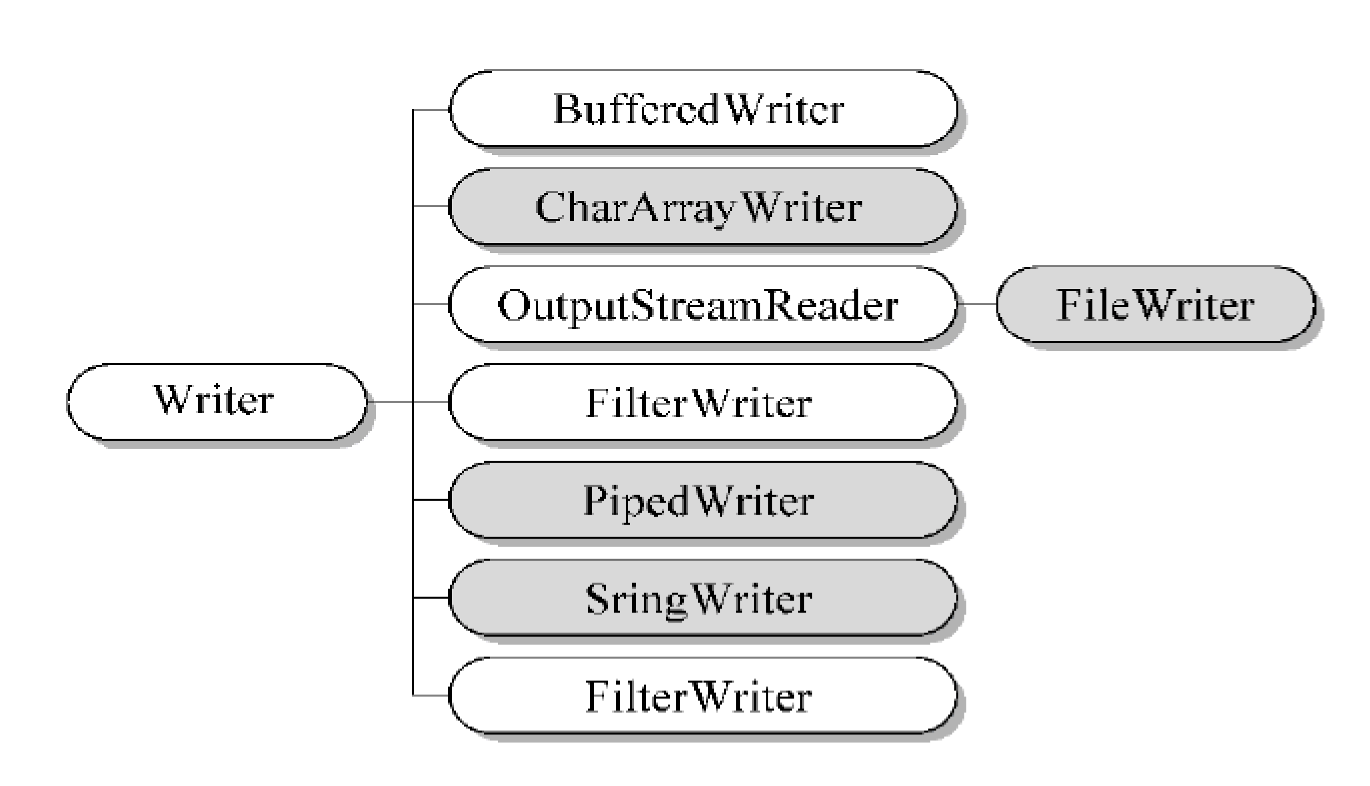


图 10-9　Writer 类结构图

是一样的，只是字符流中以字符为基本的读写单位，这里不再重复。

表 10-4　字符流常用子类及功能

字 符 流 类	功能简单介绍
StringReader	从字符串读取的输入流
StringWriter	写入字符串的输出流
FileReader	从文件读入的输入流
FileWriter	写入文件的输出流
PushbackReader	返回一个字符到输入流，主要用于编译器的实现
PipedReader	输入管道
PipedWriter	输出管道
CharArrayReader	从字符数组读取的输入流
CharArrayWriter	写入字符数组的输出流
BufferReader	缓冲输入流
BufferWriter	缓冲输出流
FilterReader	实现了 InputStream 接口的过滤输入流
FilterWriter	实现了 OutputStream 接口的过滤输出流
InputStreamReader	将字节转换为字符的输入流
OutputStreamWriter	将字节转换为字符的输出流

10.3 文件的操作

Java 提供了许多进行文件输入输出的类。从读写文件类型的不同，可以将这些类分为文本 I/O 类与二进制 I/O 类对应对文本文件和二进制文件的读写。在文本文件中存储的数据是以字符编码方式表示的，而在二进制文件中存储的数据是用二进制形式来表示的。Web 程序更多地需要对文本形式的文件进行读写，故本章只对文本文件的读写进行介绍，关于二进制形式文件的读写，读者可以查阅相关的 Java 书籍。

10.3.1 读取文件

在读取文件中，可以使用两种方式：第一种是字节流 InputStream 的方式；另一种是使用字符流 Reader 的方式。虽然两种方法都可实现对文本文件的读取，但前者是按照字节方式读取，读取后需要对字符进行重新编码，特别是中文字符，如果读取后没有重新编码，会出现乱码的问题。所以在 JSP 中，使用 Reader 会有更好的性能。

实际在 JSP 中使用的是 Reader 的子类 java.io.FileReader，同时为了提高输入效率，可以使用带缓冲(java.io.BufferedReader)的输入流方式来读取文件资源。

FileReader 的使用前文中已经介绍，这里不再重复，但应该注意如果需要读取的文件不存在，则会抛出一个 FileNotFoundException。

BufferedReader 的输入方法和表 10-3 中所示一样，只是多了一次读取一行的方法：String readLine()，该方法一次读取一行文本，返回为字符串。

程序清单 10-3 演示了一个读取文本文件 dt.txt 到页面显示的程序。

程序清单 10-3：

```
<%--readfile1.jsp--%>
<%@page contentType="text/html;charset=gb2312"%>
<%@page import="java.io.*"%>
<html>
<head>
<title>读取文件数据</title>
</head>
<body>
<%
try{
  String path=getServletContext().getRealPath(".");
  FileReader input=new FileReader(path+File.separator+"dt.txt");
  int in=input.read();                    //从文件中读取一个字符
  while(in!=-1){                          //判断是否已读到文件结尾
    out.print((char)in);
    in=input.read();
    if(in==13){                           //判断是否为断行字符
      out.print("<br>");
      input.skip(1);                      //略过一个字符
```

```
          in= input.read();
        }
      }
      input.close();
  }catch(FileNotFoundException fnfe){
      out.println(fnfe);                    //文件不存在异常
  }catch(IOException ioe){
      out.println(ioe);
  }
  %>
  </body>
  </html>
```

如果 dt.txt 不存在，则程序抛出异常，显示结果如图 10-10 所示，文件正常读取后结果如图 10-11 所示。



图 10-10　文件读取异常

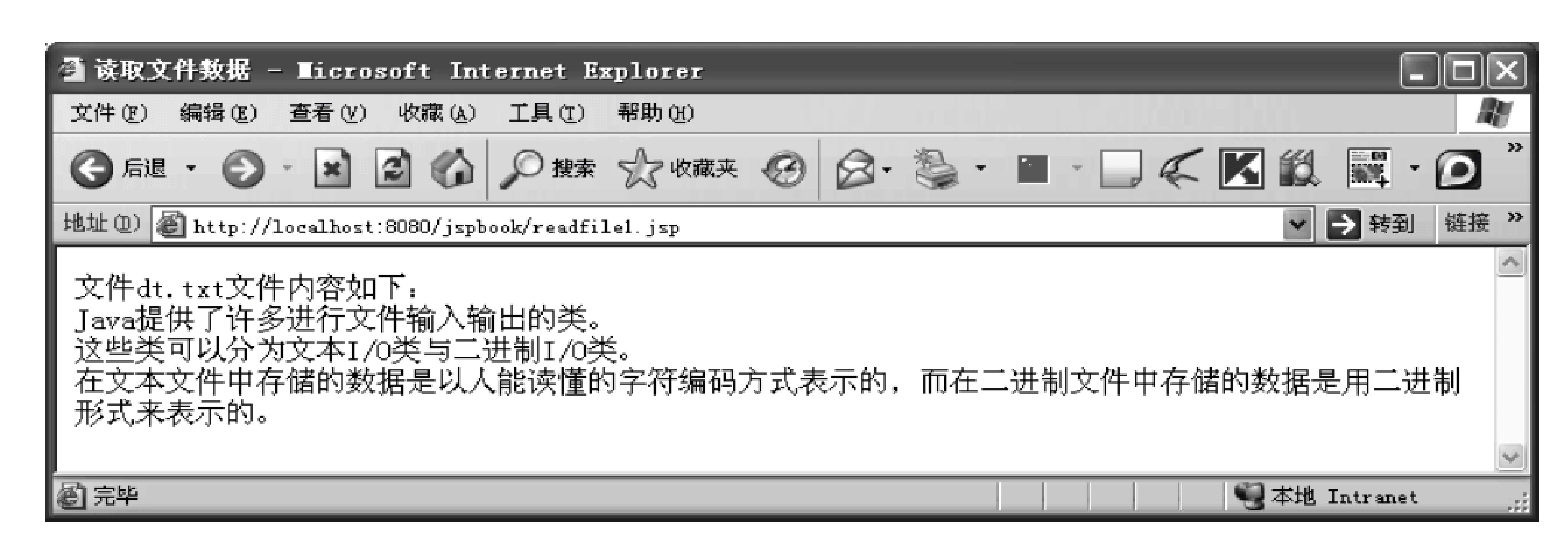


图 10-11　文件读取成功

在程序清单 10-3 中没有缓冲 BufferedReader，直接使用了 FileReader 读取文件，若要使用缓冲，则程序如程序清单 10-4，结果与图 10-11 一样。

程序清单 10-4：

```
<%--readfile2.jsp--%>
<%@page contentType="text/html;charset=gb2312"%>
<%@page import="java.io.*"%>
<html>
<head>
<title>读取文件数据</title>
```

```
</head>
<body>
<%
try{
   String path=getServletContext().getRealPath(".");
   FileReader frinput=new FileReader(path+File.separator+"dt.txt");
   BufferedReader brinput=new BufferedReader(frinput);
   String file="";
   String temp="";
   while((temp=brinput.readLine())!=null){    //读一行并判断是否已读到文件结尾
      file+=temp+"<br>";                       //realLine 本身不返回换行符
   }
   out.print(file);
   brinput.close();
   frinput.close();
}catch(FileNotFoundException fnfe){
   out.println(fnfe);
}catch(IOException ioe){
   out.println(ioe);
}
%>
</body>
</html>
```

10.3.2 写入文件

在写入文件中，同样可以使用字节流和字符流两种方式，这里介绍使用字符输出流 Writer 的子类 java.io.FileWriter。也可以使用缓冲(java.io.BufferedWriter)的方法来提高输出效率。

FileWriter 的使用前文中已经介绍，这里不再重复，但应该注意如果指定文件存在，但它是一个目录，而不是一个常规文件；或者该文件不存在，但无法创建它；或因为其他某些原因而无法打开它，则会抛出一个 IOException。

BufferedWriter 中的 write()除了和表 10-3 中一样的输出外，还支持一次输出一个字符串的方式 write(String)，newLine()方法实现向文件写入一个依赖平台的换行符。缓冲区输出只有当缓冲区已满或者调用 flush()方法时才调用写入文件的方法。

程序清单 10-5 演示将从 dt.txt 读取的文本内容写入 newdt.txt 的程序，运行后将会生成一个新的 newdt.txt 文件，并将 dt.txt 中的内容写入 newdt.txt。

程序清单 10-5：

```
<%--writefile1.jsp--%>
<%@page contentType="text/html;charset=gb2312"%>
<%@page import="java.io.*"%>
<html>
<head>
```

```
<title>复制文件数据</title>
</head>
<body>
<%
try{
   String path=getServletContext().getRealPath(".");
   FileReader fr=new FileReader(path+File.separator+"dt.txt");
   BufferedReader brinput=new BufferedReader(fr);
   FileWriter wf=new FileWriter(path+File.separator+"newdt.txt");
   BufferedWriter bwoutput=new BufferedWriter(wf);
   String temp="";
   while((temp=brinput.readLine())!=null){
      bwoutput.write(temp);
      bwoutput.newLine();                    //输出一个换行符
   }
   brinput.close();
   bwoutput.close();
   wf.close();
   fr.close();
}catch(FileNotFoundException fnfe){
   out.println(fnfe);
}catch(IOException ioe){
   out.println(ioe);
}
%>
</body>
</html>
```

程序清单 10-6 实现了在 Web 页中嵌入一个文本框，并将文本框中的内容写入到了文件 dt.txt 中。文本框用 html 表单<form>中的<textarea>来表示。运行结果如图 10-12 所示。

程序清单 10-6：

```
<!--writefile2.jsp-->
<%@page contentType="text/html; charset=gb2312"%>
<%@page import="java.io.*"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
</head>
<body>
<%
String action=request.getParameter("action");
if(action!=null){
   String content=new String(
```

```
            request.getParameter("content").getBytes("iso-8859-1"),"gb2312");
    try{
      String path=getServletContext().getRealPath(".")+File.separator+"dt.txt";
      BufferedWriter bfwriter=new BufferedWriter(new FileWriter(path));
      bfwriter.write(content);
      bfwriter.close();
      out.println("已经把内容写入到 dt.txt");
    }catch(Exception e){
      out.println(e);
    }
  }
%>
<form action="writefile2.jsp?action=submit" method="post">
<table>
<tr><td><textarea name="content" cols="40" rows="10"></textarea></td></tr>
<tr><td><input type="submit" value="提交"></td></tr>
</table>
</form>
</body>
</html>
```

程序清单 10-6 中 writefile2.jsp 文件将文本框内容提交到 writefile2.jsp?action=submit，其中 action=submit 是通过 QueryString 告诉文件是否是由表单提交的内容。语句 new String(request.getParameter("content").getBytes("iso-8859-1"),"gb2312");是由于网页默认采用"iso-8859-1"发送文本内容，这样中文字符就会显示乱码，该语句将提交内容由 iso-8859-1 编码转为 gb2312 编码格式，目的是正常显示汉字字符。程序运行结果如图 10-12 所示。



图 10-12 将文本内容写入文本文件中

10.3.3 追加操作

程序清单 10-6 已经实现将文本内容写入文件 dt.txt，但是上面的代码有一个问题，每次程序执行写入的新内容把原来文件中的老内容覆盖了，无法实现文件的追加。若要实现对文件的追加可以有两种方法，一种是使用带有 boolean 参数的 FileWriter 构造方法，另一种是使用随机读写文件类 RandomAccessFile。RandomAccessFile 将在下一节介绍，在这里简单介绍第一种方法。

FileWriter 有带有 boolean 参数的构造方法：

(1) FileWriter(File file, boolean append)。

(2) FileWriter(String fileName, boolean append)。

其中，第二个参数 boolean append，如果为 true，则将数据写入文件末尾处，而不是写入文件开始处。

所以要实现文件的追加，只要在程序清单 10-6 中改变如下代码即可：

```
String path=request.getRealPath(".")+File.separator+"dt.txt";
BufferedWriter bfwriter=new BufferedWriter(new FileWriter(path, true));
```

10.3.4 使用 RandomAccessFile 类

使用 FileWriter 构造方法的第二个参数 boolean append 只能设定文件写入位置是在文件开头还是末尾，而用 RandomAccessFile 实现文件灵活随机读写更加方便，RandomAccessFile 的实例支持对随机存取文件的读取和写入。随机存取文件类似存储在文件系统中的一个大型字节数组。存在一个指向该隐含数组的光标或索引，称为“文件指针”。输入操作从文件指针开始读取字节，并随着对字节的读取而前移动此文件指针。如果随机存取文件以读取/写入模式创建，则输出操作时，输出操作从文件指针开始写入字节，并随着对字节的写入而前移此文件指针。写入隐含数组的当前末尾之后的输出操作导致该数组扩展。该文件指针可以通过 getFilePointer 方法读取，并通过 seek 方法设置。

RandomAccessFile 的构造方法有两个：

(1) RandomAccessFile(File file,String mode)。

(2) RandomAccessFile(String name,String mode)。

这两个构造方法的第一个参数用于指定需要打开的文件，而第二个参数 String mode 表示打开文件的方式，共有以下 4 个参数可选。

(1) “r”：以只读方式打开，调用任何 write 方法都将导致抛出 IOException。

(2) “rw”：以读取和写入打开，如果该文件不存在，则尝试创建该文件。

(3) “rws”：以读取和写入打开，还要求对文件的内容或元数据的每个更新都同步写入到基础存储设备，这里的元数据指文件的信息数据，如最后修改时间，长度等。

(4) “rwd”：以读取和写入打开，还要求对文件内容的每个更新都同步写入到基础存储设备，但不写入元数据的每个更新。

“rws”和“rwd”可以保证对此文件所做的所有更改均被及时写入相应设备，这对确保在系统崩溃时不会丢失重要信息特别有用，但是这会增加 I/O 操作的次数。

程序清单 10-7 演示了用 RandomAccessFile 实例进行随机读写的例子。运行结果如图 10-13 所示。

程序清单 10-7：

```
<!--writefile3.jsp-->
<%@page contentType="text/html; charset=gb2312"%>
<%@page import="java.io.*"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>无标题文档</title>
</head>
<body>
<%
String action=request.getParameter("action");
String text="";
String content="";
if(action!=null){
  content=new String(
          request.getParameter("content").getBytes("iso-8859-1"),"gb2312");
  try{
    String path=getServletContext().getRealPath("."+File.separator+"dt.txt");
    //定义一个类 RandomAccessFile 的可读写对象将 content 写入文件
    RandomAccessFile rafwriter=new RandomAccessFile(path,"rw");
    long n=rafwriter.length();
    rafwriter.seek(n);
    rafwriter.write(content.getBytes());
    rafwriter.close();
    //定义一个类 RandomAccessFile 的只读对象将本次输入读出
    String temp="";
    RandomAccessFile rafreader=new RandomAccessFile(path,"r");
    rafreader.seek(n);
    while((temp=rafreader.readLine())!=null){
      text+=temp+"<br>";
    }
    rafreader.close();
  }catch(Exception e){
    out.println(e);
  }
}
%>
<form action="writefile3.jsp?action=submit" method="post">
<table>
<tr>
<td><textarea name="content" cols="40" rows="10"></textarea></td>
```

```
<td><p>本次追加的内容为:</p><%=content%></td>
</tr>
<tr><td><input type="submit" value="提交"></td></tr>
</table>
</form>
</body>
</html>
```



图 10-13　RandomAccessFile 读写文件

10.4　文件上传

要进行文件上传,可以用设计一个 JavaBean 或者 Servlet 处理上传任务,基本思路是由 JavaBean 或者 Servlet 获取客户端请求的输入流,然后从这个输入流中读取文件内容,然后把这些内容保存为相应的文件。说起来很简单,但实际处理确实比较复杂。

所以,在这里向读者介绍一个免费的上传组件 jspSmartUpload,适用于嵌入执行上传操作的 JSP 文件中。该组件使用简单,在 JSP 文件中仅仅只需要书写三五行代码就可以实现文件的上传,并能够全程控制上传,利用 jspSmartUpload 组件提供的对象及其操作方法,可以获得全部上传文件的信息(包括文件名,大小,类型,扩展名,文件数据等),方便存取,而且能对上传的文件在大小、类型等方面做出限制,可以滤掉不符合要求的文件。下面对 jspSmartUpload 组件的安装和使用做一个简单的介绍。

jspSmartUpload 组件可以免费下载,下载后文件的名字是 jspSmartUpload.zip。解压后只需要将 web-inf 目录下的文件复制到 Web 应用程序所在目录的 WEB-INF 文件夹下就可以在 Web 程序中使用上传组件了。如果想让服务器的所有 Web 应用程序都能用它,将 web-inf/classes 中的 com 文件夹打包为 jspSmartUpload.jar 并复制到服务器的 lib 目录下即可。

jspSmartUpload 包含有 5 个类。

(1) File:上传文件的抽象表示类。

(2) Files:包含多个上传文件的 File 实例。

（3）Request：等价于 Servlet 的 ServletRequest 类。

（4）SmartUpload：实现上传的类。

（5）SmartUploadException：上传抛出的异常类。

在实际上传处理中，一定要用到的是 File 类和 SmartUpload 类，File 实例可以调用 Files 的 getFile(int)方法创建，而 Files 可以调用 SmartUpload 的 getFiles()创建，如下语句：

```
SmartUpload mySmartUpload=new SmartUpload();
File f=mySmartUpload.getFiles().getFile(0);  //得到上传的第一个文件实例
```

File 的常用方法见表 10-5 所示。利用 SmartUpload 类实现上传功能，可以创建 JavaBean 或者直接在 JSP 文件中创建实例，SmartUpload 的常用方法见表 10-6 所示。

表 10-5　File 的常用方法

方　　法	功　　能
void saveAs(String)	保存文件到指定的目录
void saveAs(String, int)	保存文件到指定的目录，其中第二个参数有 3 个常量可选： （1）SAVEAS_PHYSICAL 表明以操作系统的根目录为文件根目录另存文件； （2）SAVEAS_VIRTUAL 表明以 Web 应用程序的根目录为文件根目录另存文件； （3）SAVEAS_AUTO 则表示让组件决定
String getFileExt()	得到文件扩展名
String getFileName()	得到文件名
String getFilePathName()	得到文件的完整路径
int getSize()	得到文件的长度
boolean isMissing()	判断上传文件是否丢失

表 10-6　SmartUpload 的常用方法

方　　法	功　　能
Files getFiles()	得到上传的文件列表
int getSize()	得到上传文件的总长度
void initialize(PageContext)	用内嵌的 pagecontext 对象初始化
int save(String)	保存所有的上传文件到指定目录
void setAllowedFilesList(String)	设置允许上传的文件类型
void setDeniedFilesList(String)	设置不允许上传的文件类型
void setMaxFileSize(long)	设置最大的上传文件长度
void setTotalMaxFileSize(long)	设置上传文件的总长度
void upload()	从表单选择上传文件

文件的上传使用表单<form>，对于上传文件的<form>表单，有两个要求：

(1) method 属性应该使用 post 方法；

(2) 增加属性：enctype="multipart/form-data"，表示采用二进制的方式上传。

程序清单 10-8 是一个支持一次上传 4 个文件的表单，效果如图 10-14 所示。



图 10-14　上传表单

程序清单 10-8：

```
<!--upload.htm-->
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>文件上传</title>
</head>
<body>
<h1>jspSmartUpload 文件上传</h1>
<hr>
<form method="post" action="upload.jsp" enctype="multipart/form-data">
  <table border="0" cellspacing="0" cellpadding="0">
    <tr><td>第一个文件</td>
        <td><input name="file1" type="file" size="40"></td>
    </tr>
    <tr><td>第二个文件</td>
        <td><input name="file2" type="file" size="40"></td>
    </tr>
    <tr><td>第三个文件</td>
        <td><input name="file3" type="file" size="40"></td>
    </tr>
    <tr><td>第四个文件</td>
        <td><input name="file4" type="file" size="40"></td>
    </tr>
```

```
    <tr><td colspan="2"><input type="submit" name="Submit" value="上传"></td>
    </tr>
  </table>
</form>
</body>
</html>
```

表单将文件提交到 upload.jsp 文件，程序清单 10-9 是 upload.jsp 文件，上传结果如图 10-15 所示。

程序清单 10-9：

```
<%--upload.jsp--%>
<%@page contentType="text/html; charset=gb2312"%>
<%@page import=" com.jspsmart.upload.* "%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>上传文件列表</title>
</head>
<jsp: useBean id="mySmartUpload"
scope="page" class="com.jspsmart.upload.SmartUpload"/>
<body>
<%
try{
  int count=0;
  mySmartUpload.initialize(pageContext);
  //初始化
  mySmartUpload.setMaxFileSize(10000);
  //限制上传文件的长度
  mySmartUpload.setTotalMaxFileSize(20000);
  //限制上传文件的总长度
  mySmartUpload.setAllowedFilesList("doc,txt,bak,html,htm,jsp");
  //设定允许上传的文件
  mySmartUpload.setDeniedFilesList("exe,bat");  //设定禁止上传的文件
  mySmartUpload.upload();                       //上传
  out.println("<table border=1>");
  out.println("<tr><th>表单项名</th><th>文件名</th><th>文件长度</th>");
  out.println("<th>文件扩展名</th><th>文件的完整路径</th></tr>");
  for(int i=0;i<mySmartUpload.getFiles().getCount();i++){
    File myFile=mySmartUpload.getFiles().getFile(i);
    if (!myFile.isMissing()) {
      myFile.saveAs("/upload/"+myFile.getFileName(),
SmartUpload.SAVE_VIRTUAL);
  out.println("<tr>");
  out.println("<td>"+myFile.getFieldName()+"</td>");
  out.println("<td>"+myFile.getFileName()+"</td>");
```

```
        out.println("<td>"+myFile.getSize()+"</td>");
        out.println("<td>"+myFile.getFileExt()+"</td>");
        out.println("<td>"+myFile.getFilePathName()+"</td>");
        out.println("</tr>");
        count++;
          }
        }
        out.println("</table>");
        out.println("<p>应上传"+mySmartUpload.getFiles().getCount()+"个文件</p>");
        out.println("<p>实际上传"+count+"个文件,上传总长度为"
    +mySmartUpload.getSize()+"</p>");
    }catch(Exception e){
        out.println(e);
    }
    %>
    </body>
    </html>
```



表单项名	文件名	文件长度	文件扩展名	文件的完整路径
file	LICENSE.txt	2532	txt	E:\PData\Jspbook\LICENSE.txt
file2	sample1.htm.bak	445	bak	E:\PData\Jspbook\sample1.htm.bak
file3	sample2.htm	425	htm	E:\PData\Jspbook\sample2.htm
file4	readfile1.jsp	772	jsp	E:\PData\Jspbook\readfile1.jsp

应上传4个文件

实际上传4个文件，上传总长度为4995

图 10-15　上传文件列表

小　　结

JSP 的文件操作是非常重要的内容，在这一章中介绍了利用 java.io.File 类获取文件属性的方法，介绍了 JSP 中的输入流和输出流，介绍了文件读写的操作，最后介绍了文件上传的方法。

java.io.File 类包含许多获取文件和目录属性的方法以及创建、删除和重命名文件和目录的方法，在 JSP 中可以使用 File 对文件和文件夹进行管理。

为了对文件读写，必须使用 JSP 的输入流和输出流，JSP 使用 Java I/O 系统，所以它的输入、输出流和 Java 的基本一致。本章向读者介绍了如何使用 java.io.FileReader 和 java.io.BufferedReader 配合实现文本文件的缓冲输入，介绍了如何使用 java.io.FileWriter 和

java. io. BufferedWriter 配合实现文本文件的缓冲输出，还介绍了通过 java. io. RandomAccessFile 实现文件的随机灵活读写。

最后本章介绍了通过 jspSmartUpload 组件实现文件的上传，jspSmartUpload 是一个免费的 JavaBean 组件，可以嵌入到 JSP 页面中简单地实现文件的上传与上传文件的管理和追踪。

通过本章的学习，读者应该可以掌握 JSP 中文件的获取属性、读写、追加、上传等基本操作。

练　习　10

1. 填空题

(1) java. io. File 中获得文件名、规范路径、绝对路径、文件长度的方法分别是________、________、________和________。

(2) File 类有 4 个常量：pathSeparator、pathSeparatorChar、separator 和 separatorChar 分别表示________、________、________和________。

(3) 在 Windows 环境中，创建一个 c：\dt. txt 的 File 对象应该用________语句；创建一个表示当前目录下 dt. txt 的 File 对象应该用________语句。

(4) 表示 Java I/O 中基本抽象输入/输出字节流和字符流的分别是________、________、________和________。

(5) 创建 RandomAccessFile 的 4 种模式"r"、"rw"、"rws"和"rwd"分别表示________、________、________和________。

2. 选择题

(1) 下列 FileReader 类中________方法可以用于关闭流。

A. skip()　　B. close()　　C. mark()　　D. reset()

(2) 下面代码将 f1. txt 的内容复制到 f2. txt 中，完成如下程序：

```
try{
    ____①____ ;
    FileReader fin=new FileReader("f1.txt");
    FileWriter fout=new FileWriter("f2.txt");
    int c;
    while((c=fin.read())!=-1){
        ____②____ ;
        str.append((char)c);
    }
    fin.close();
    fout.close();
}catch(Exception c){}
```

① A. String str=new String ()

B. StringBuffer str=new StringBuffer()

C. StringTokenizer str=new StringTokenizer()

D. char[] str=new char[256]

② A. fout.read()　　B. fout.print()

C. fout.print (c)　　D. fout.write(c)

3. 实验题

设计一个基于 Web 页的文本文件编辑器,可以对文件进行在线的编辑。

第 11 章　JSP 访问 Web 数据库

11.1　JDBC 简介

JSP 开发离不开数据库编程，几乎所有的 JSP 开发的 Web 程序都和数据库有关，JSP 中提供了通过 JDBC 接口访问数据库的方法，允许开发人员通过几句简单的代码连接访问数据库，这大大简化了 JSP 数据库开发的难度。从本节开始，向读者介绍 JSP 是如何访问数据库的。

11.1.1　JDBC 基本概念

JDBC 是 Java DataBase Connectivity 的简称，是一种用 Java 实现的数据库接口技术，是开放数据库 ODBC 的 Java 实现。应用程序要完成对数据库中数据的操作，通常要使用 SQL 语言中的有关语句，但是 SQL 语言是一种非过程语言，除了对数据库基本操作外，它所能完成的功能非常有限，并不能适应整个前端的应用编程。为此，需要其他的语言来实现 SQL 语言的功能以完成对数据库的操作。为了达到这个目的，JDBC 定义了 Java 语言同 SQL 数据之间的程序设计接口，它是一个非常独特的动态连接结构，通过模块化的方式完成对数据库的操作，这个接口中定义了很多用来实现 SQL 功能的类，使用这些类，编程人员就可以很方便地开发出数据库前端的应用。用 JDBC 开发数据库应用的原理如图 11-1 所示。

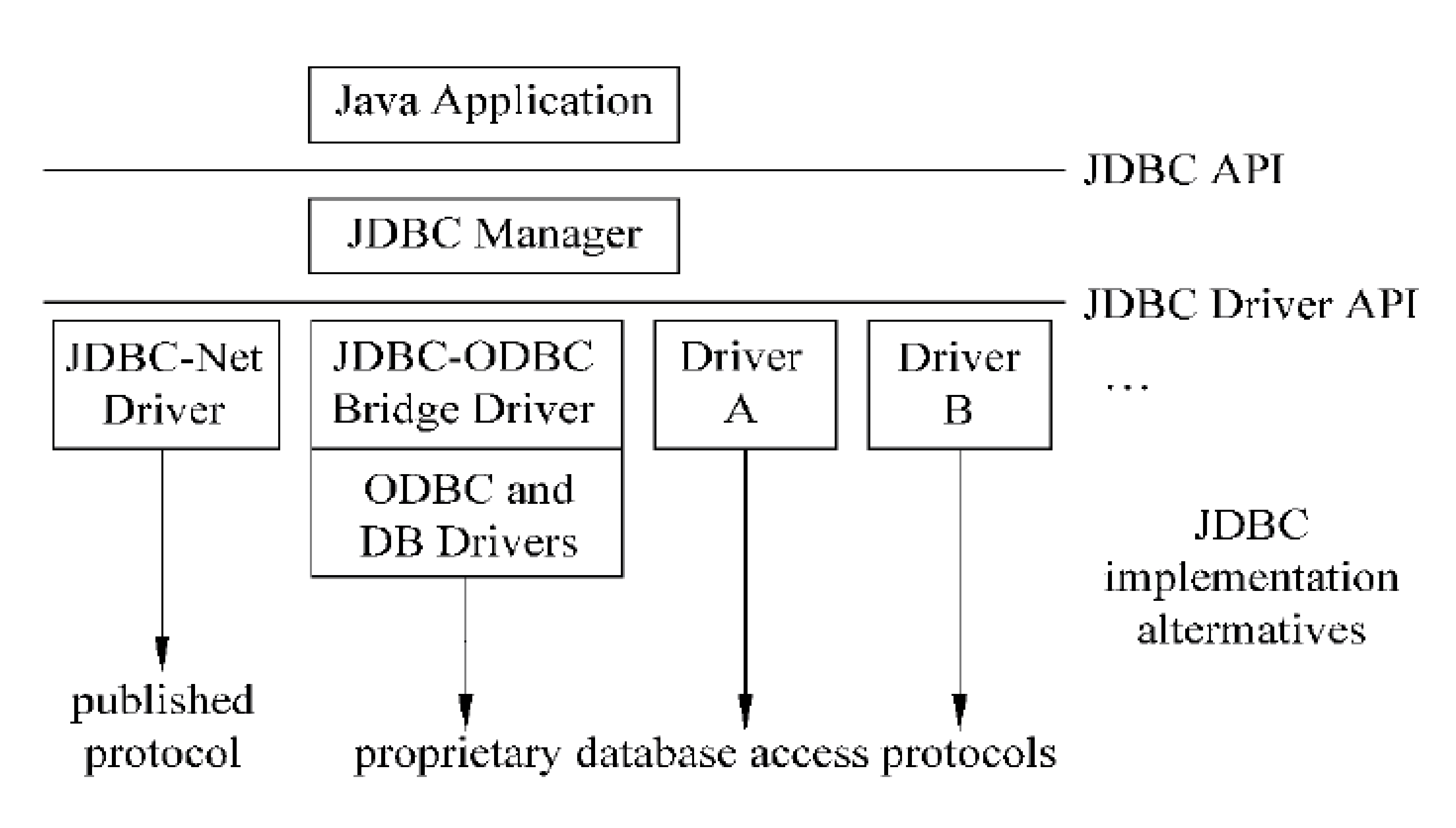


图 11-1　JDBC 工作原理

由图 11-1 可知，JDBC 由两层组成。上面一层是 JDBC API，负责与 Java 应用程序通信，向 Java 应用程序提供数据(Java 应用程序通过 JDBC 中提供的相关类来管理 JDBC 的驱动程序)。下面一层是 JDBC Driver API，主要负责具体数据环境的连接。

JDBC 技术能够快速地被人们接受并广泛地应用于各种 Java 数据库开发程序中，正是由于有以下的一些特点。

(1) 数据库开发的对象化。在 JDBC 中有两个包支持数据库开发。

① java. sql：这个包中的类和接口主要针对基本的数据库编程服务，如生成连接、执行 SQL 语句、批处理操作和事务操作。

② javax. sql：这个包为数据库开发提供了高级操作的接口和类，如连接管理、分布式事务，引入了容器管理的连接池等技术。

这些类和接口的使用可以使数据库开发完全对象化。

(2) 数据库开发的标准化。数据库开发只和上层的 JDBC API 打交道，不和数据库直接联系，这就使得数据库应用开发趋向标准化，不存在不同数据库软件的不同开发模式。

一般的 JDBC 对数据库的一次操作流程分为如下几个步骤。

① 装载 JDBC 驱动程序。

② 定义连接字符串。

③ 和数据库建立连接。

④ 创建表达式对象。

⑤ 执行数据库操作(查询，删除，更新，修改)。

⑥ 处理返回结果。

⑦ 关闭数据库连接。

(3) 支持多个关系数据库。JDBC 现在可以连接的数据库包括 Oracle、SQL Server、MySQL、Sybase、DB2、Access 等，满足不同编程人员对数据库开发的要求。

11.1.2 数据库的连接方式

总体来说，有 4 种数据库连接的方式：JDBC-ODBC 桥；部分 Java，部分数据库专用 API；中间件访问；纯 Java 驱动访问。

下面对这 4 种方式做一个简单的介绍。

(1) JDBC-ODBC 桥。在 JDBC 出现的早期，这种驱动方式是比较流行的。通过这个驱动，可以将 JDBC 对数据库的操作映射为 ODBC 对于数据库的操作，如图 11-2 所示。早期由于各个数据库厂商提供的针对各数据的 JDBC 驱动较少，为了拓展 JDBC 的应用，Sun 公司在 JDK 1.1 中提供了这种实现方式，由于 ODBC 是当今数据库连接方面比较流行的解决方案，这种桥接方式的驱动减少了用户采用 Java 解决方案时的顾虑。

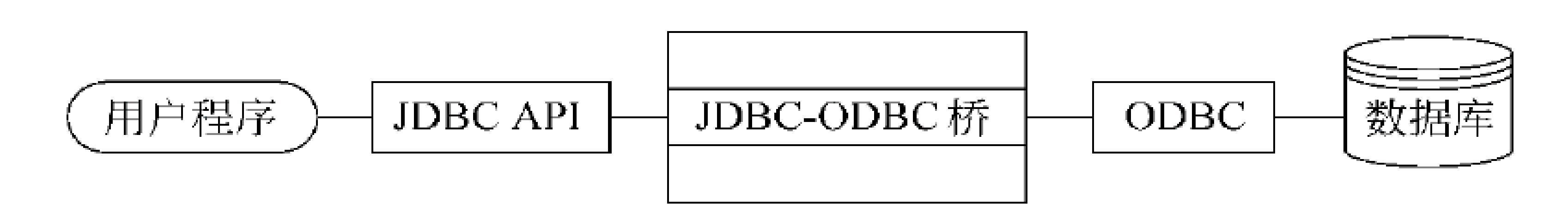


图 11-2　JDBC-ODBC 桥

但使用这种方法的缺点是显而易见的，由于 ODBC 驱动是由 C 语言开发的，仅仅支持 Windows 操作系统，因此使用这种桥接的方式将 Java 的数据库应用限制在了 Windows 平台下，使其失去了平台无关性的特点。同时，由于需要在两种接口间互相调用，执行的效率相对于其他直接驱动方式来说是比较低的，所以现在这个方式应用较为少见。

(2) 部分 Java，部分数据库专用 API。这种方式使用 Java 实现和数据库专用 API 混合方式来连接。JDBC 驱动将标准的 JDBC 调用转化为对数据库 API 的本地调用，这种方式

使用的驱动程序是部分 Java 驱动和本地 API 驱动程序，如图 11-3 所示。

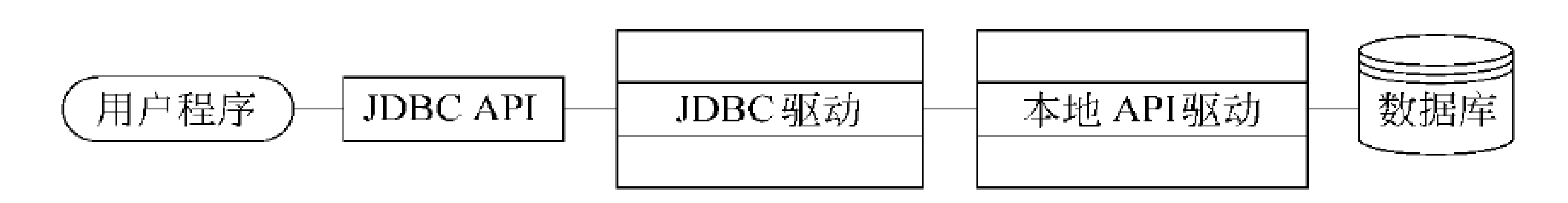


图 11-3　部分 Java，部分数据库专用 API

和 JDBC-ODBC 桥驱动程序一样，这种类型的驱动程序要求将某些二进制代码加载到每台客户机上，因此也将失去平台无关性的好处。现在大多数数据库厂商都会为其数据库产品提供该类驱动程序，在性能上，这种驱动方式比 JDBC-ODBC 桥方式要好。

(3) 中间件访问。这种访问方式将 JDBC 的操作指令转换为驱动程序厂商自己定义的网络协议，该网络协议是与 DBMS 无关。通过协议访问某种特定的中间件服务器。这个特定的中间件服务器往往位于 Web 服务器或者数据库服务器上。然后由这个中间件服务器将网络协议再转换为某种特定的 DBMS 协议，来调用数据库。数据库处理的结果也将按照这个相反的过程返回给 Java 程序端。

由于这种方式用到的驱动程序是纯 Java 的，这样就可以在客户端使用一个纯 Java 的程序。这种 JDBC 方案比前两种更具有弹性，而且很适合用做基于 Web 的应用。其基本工作过程如 11-4 所示。

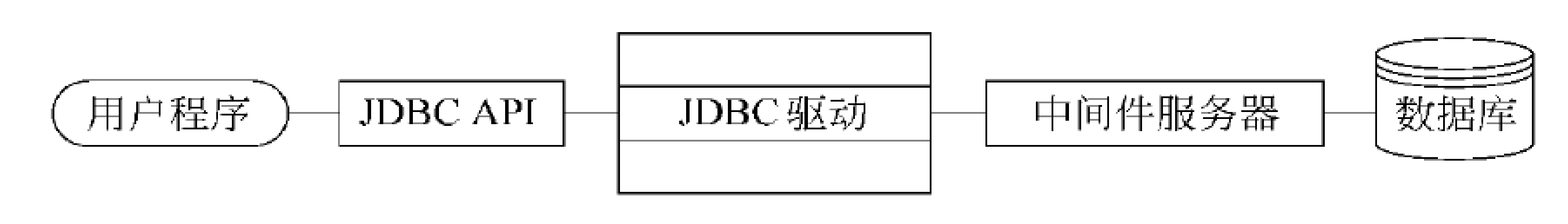


图 11-4　中间件访问

(4) 纯 Java 驱动访问。这种访问方式使用厂商专用的驱动程序把 JDBC 对数据库的操作直接转换为针对某种数据库进行操作的本地协议。与其他类型的驱动相比，由于它在调用过程中不再需要像 JDBC 的网络驱动一样转化为其他的形式，本质上是一种类似于 Socket 的连接方式，因此在执行效率上很有优势，其工作过程如图 11-5 所示。

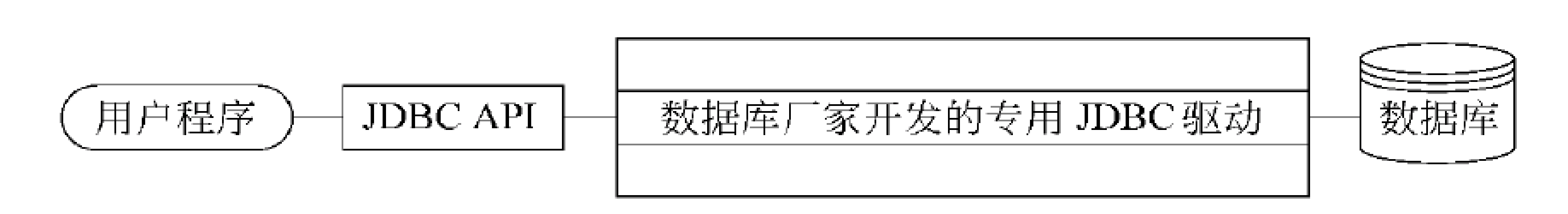


图 11-5　纯 Java 驱动访问

前两种方案由于失去了 JDBC 的平台无关性，使用的范围并不是很广，主要作为应急手段或者初期开发方式出现，而一般来说后两种方案是现在的主流的 JDBC 数据库连接方式。后两种方案采用纯 Java 的方式连接数据库，这样可以保证程序的跨平台性，便于程序发布，而且现在主流的数据库厂商都提供了专用的 JDBC 驱动程序，所以读者应该尽可能在学习过程中就使用后两种方案。下面通过一个简单的例子来演示如何在 JSP 程序中连接数据库，见程序清单 11-1，在这个程序中使用的是 MySQL 数据库。

程序清单 11-1：

```
<!--testconn.jsp-->
```

```
<%@page contentType="text/html;charset=gb2312"%>
<%@page import="java.sql.*"%>
<html>
<head>
<title>数据库连接测试</title>
</head>
<body>
<%
try{
  Class.forName("com.mysql.jdbc.Driver").newInstance();
  String url ="jdbc: mysql: //localhost/test";
  Connection conn=DriverManager.getConnection(url,"root","");
  if(conn!=null)
    out.print("<h3>数据库连接成功,恭喜你</h3>");
  Statement stmt=conn.createStatement();
  String sql="select user(),now()";
  ResultSet rs=stmt.executeQuery(sql);
  while(rs.next()){
    out.print("当期数据库用户为:"+rs.getString(1)+"<br>");
    out.print("当前时间为:"+rs.getString(2)+"<br>");
  }
  rs.close();
  stmt.close();
  conn.close();
} catch(ClassNotFoundException cnfe){
  out.print(cnfe);
} catch(SQLException sqle){
  out.print(sqle);
}catch(Exception e){
  out.print(e);
}
%>
</body>
</html>
```

在程序清单 11-1 中,首先装载了 MySQL 数据库服务器的 JDBC 驱动程序,然后通过连接字符串"jdbc:mysql://localhost/test"建立一个 Connection 对象,这就建立了和 MySQL 数据库服务器中 test 数据库的连接。在这个例子中还通过 select 语句查询了登录用户和当前系统时间,并显示在页面上,运行结果如图 11-6 所示。

当然在运行这个程序前,需要安装 MySQL 数据库软件,并启动 MySQL 数据库服务,还需要下载和安装 MySQL 的 JDBC 驱动程序。下面以 3 种常用的数据库服务器为例给读者介绍安装配置方法。

(1) MySQL 数据库。本文使用的是 MySQL 数据库,由于 MySQL 数据库可以免费下载,读者需要先到 http://www.mysql.com/ 去下载并安装 MySQL 数据库软件,安装完成后需要启动 MySQL 数据库服务。安装步骤读者可以查询相关的技术文档,这里不再介绍。在安装完成数据库后,还需下载和安装 JDBC 驱动,这个驱动的名称为 mysql-connector-java-x.x.x-bin.jar,其中 x.x.x 是 MySQL 数据库的版本号。MySQL 驱动的安装只需要将

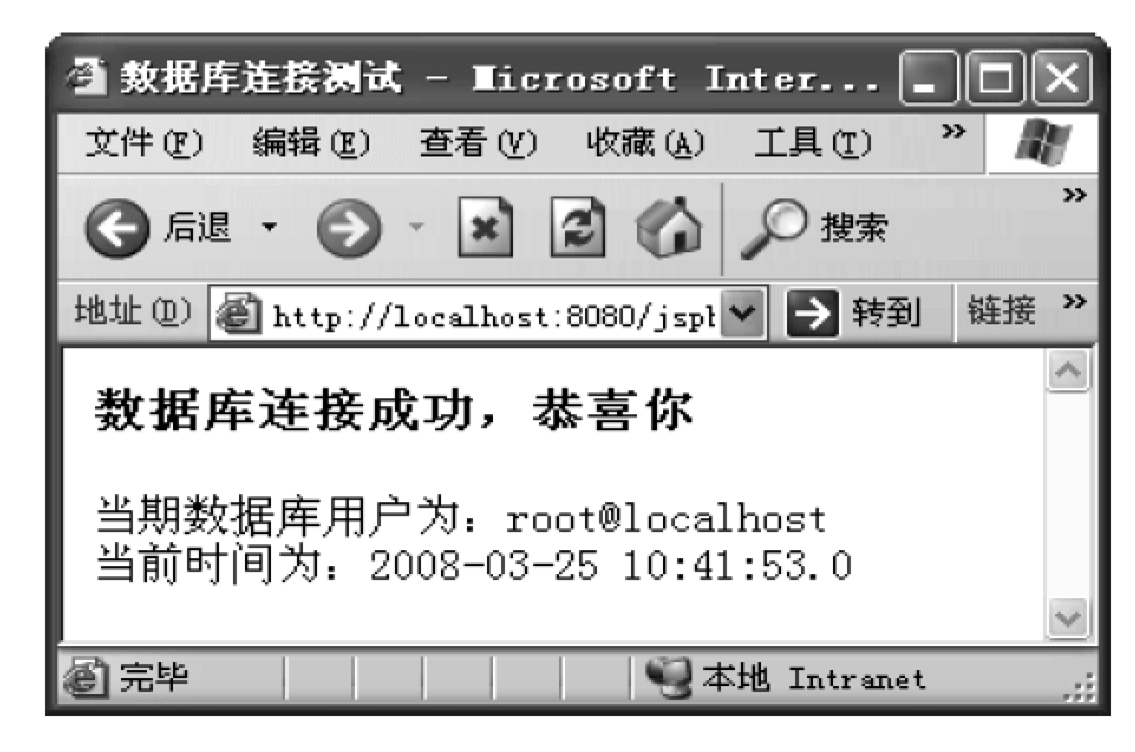


图 11-6　数据库连接示例

mysql-connector-java-x. x. x-bin. jar 复制到 Web 应用目录的/WEB-IFN/lib 中即可，当然这样只能支持在本 Web 应用中使用，若要支持所有的 Web 应用，可以将其复制到 Tomcat 安装目录的/lib/中(以 Tomcat 服务器为例)。

接下来就可以在 JSP 页码中装载这个驱动程序，并连接数据库连接，读者应该记住对应的驱动程序名称和连接字符串格式，在 MySQL 中为如下格式：

```
Class.forName("com.mysql.jdbc.Driver").newInstance();  //装载驱动程序
String url = "jdbc: mysql: //localhost/数据库名";        //建立连接字符串
Connection conn= DriverManager.getConnection(url,"root","");
      //建立连接,其中第一个参数为连接字符串,第二个为用户名,第三个为密码
```

(2) MS SQL Server 数据库。SQL Server 数据库的驱动程序有 3 个：mssqlserver. jar、msbase. jar 和 msutil. jar。

连接格式如下：

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
String url= "jdbc: microsoft: sqlserver: //localhost: 1433;DatabaseName= 数据库名";
Connection conn= DriverManager.getConnection(url,user,password);
```

(3) Oracle 数据库。Oracle 数据库的驱动程序是 classes12. jar。

连接格式如下：

```
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
String url= "jdbc: oracle: thin: @ localhost: 1521: 数据库名";
Connection conn= DriverManager.getConnection(url,user,password);
```

11. 1. 3　JDBC 常用接口

JDBC 定义了很多的接口和类，在程序清单 11-1 已经用到了 DriverManager、Connection、Statement 和 ResultSet。在 JDBC API 中对数据库的应用主要是对 DriverManager、Connection、Statement 和 ResultSet 这几个类和接口的使用，它们的调用结构如图 11-7 所示，在本节中，将对这几个类和接口做一个简要的说明。

1. DriverManager 类

DriverManager 类是 JDBC 的管理层，作用于用户和驱动程序之间。它跟踪可用的驱动

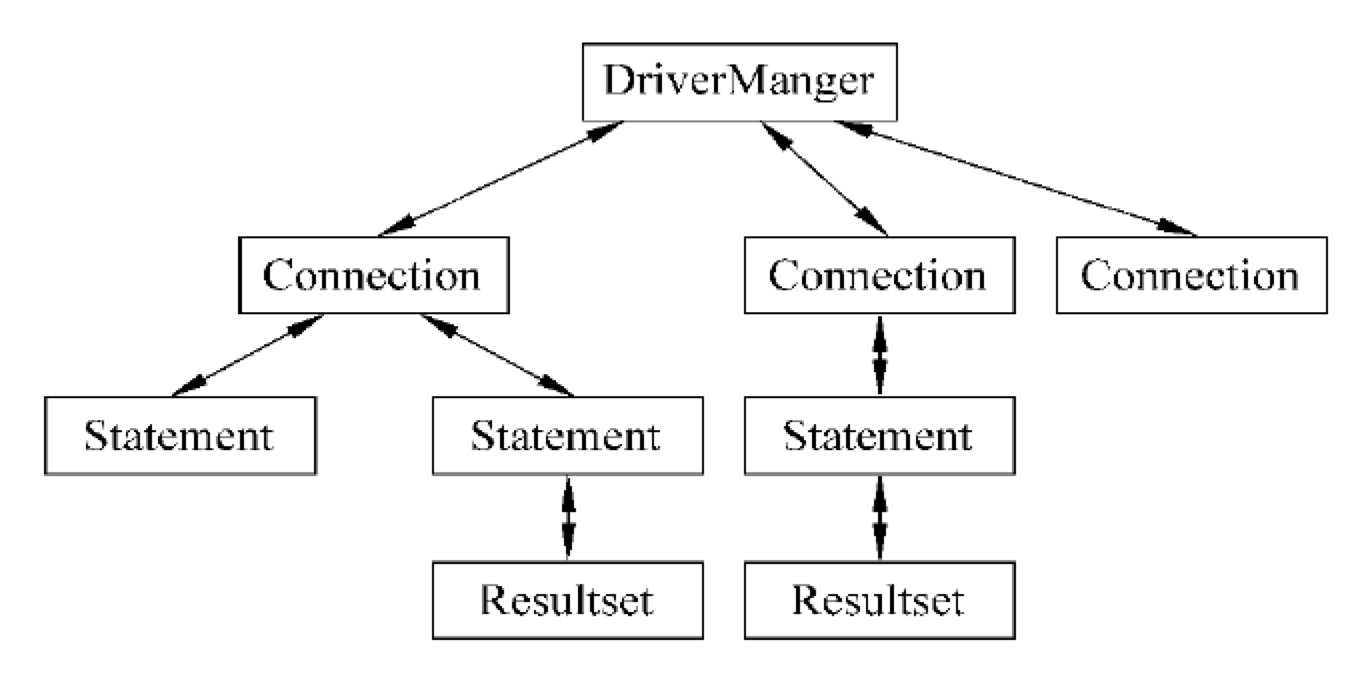


图 11-7　JDBC 主要接口的调用结构

程序,并在数据库和相应驱动程序之间建立连接。

DriverManager 类的主要方法见表 11-1。

表 11-1　DriverManager 主要方法

方　　法	功　　能
Connection getConnection(String)	试图建立到给定数据库连接字符串的连接,若不成功抛出 SQLException
Connection getConnection(String, String, String)	试图建立到给定数据库连接字符串、用户名和密码的连接,若不成功抛出 SQLException
Driver getDriver(String)	在 DriverManager 已经注册过的所有驱动中寻找能够正确访问给定连接字符串的驱动
Enumeration getDrivers()	检索带有当前调用方可以访问的所有当前已加载 JDBC 驱动程序
void deregisterDriver(Driver)	取消指定驱动在 DriverManager 类中的注册
void registerDriver(Driver driver)	向 DriverManager 类中注册某驱动

DriverManager 类包含一系列 Driver 类,它们已通过调用方法 DriverManager.registerDriver 对自己进行了注册。用户在正常情况下将不会需要直接调用 registerDriver 方法,而是在加载驱动程序时由驱动程序自动调用。用户只需要调用方法 Class.forName 显式地加载驱动程序类,然后自动在 DriverManager 类中注册,如下代码所示:

```
try{
  Class.forName("com.mysql.jdbc.Driver").newInstance(); //装载 MySQL 驱动
  Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver ").newInstance();
  //装载 SQL Server 驱动
  Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
                                              //装载 Oracle 驱动
}catch(ClassNotFoundException e){
  //若装载失败,则抛出 ClassNotFoundException
}
```

同时,该类还提供方法 getConnection()来建立与数据库的连接。此外,DriverManager 类也处理诸如驱动程序登录时间限制及登录和跟踪消息的显示等工作。

2．Connection 接口

Connection 是与数据库的连接对象，也就是在已经加载的 Driver 和数据库之间建立连接。一个应用程序可与单个数据库有一个或多个连接，或者可与许多数据库有多个连接。打开连接对象与数据库建立连接的标准方法是调用 DriverManager. getConnection()方法。该方法的参数包含有某个连接字符串。DriverManager 类将尝试找到可与该连接字符串所代表的数据库进行连接的驱动程序。DriverManager 类保存着已注册的 Driver 类的清单。当调用方法 getConnection()时，它将检查清单中的每个驱动程序，直到找到可与连接字符串中指定的数据库进行连接的驱动程序为止，代码格式如下：

```
try{
  //根据 URL 建立连接
  Connection conn= DriverManager.getConnection(url,user,password);
}catch(SQLException e){
  //若建立连接失败,则抛出 SQLException
}
```

Connection 接口的主要方法见表 11-2。

表 11-2　Connection 主要方法

方　　法	功　　能
void close()	释放此 Connection 对象的数据库和 JDBC 资源
void commit()	使自从上一次提交/回滚以来进行的所有更改成为持久更改
Statement createStatement()	创建一个 Statement 对象来将 SQL 语句发送到数据库
boolean isClosed()	检查此 Connection 对象是否已经被关闭
boolean isReadOnly()	检查此 Connection 对象是否处于只读模式
CallableStatement prepareCall(String)	创建一个 CallableStatement 对象来调用存储过程
PreparedStatement prepareStatement(String)	创建一个 PreparedStatement 对象来将参数化的 SQL 语句发送到数据库
void rollback()	取消在当前事务中进行的所有更改
void setReadOnly(boolean)	将此连接设置为只读模式

其中的 createStatement()方法，可以设定两个参数，如：createStatement(int resultSetType，int resultSetConcurrency)，这两个参数的说明如下：

```
resultSetType 值:
    TYPE_FORWARD_ONLY 结果集不可滚动
    TYPE_SCROLL_INSENSITIVE 结果集可滚动,不反映数据库的变化
    TYPE_SCROLL_SENSITIVE 结果集可滚动,反映数据库的变化
resultSetConcurrency 值
    CONCUR_READ_ONLY 不能用结果集更新数据
    CONCUR_UPDATABLE 能用结果集更新数据
```

3. Statement 接口

Statement 用于将 SQL 语句发送到数据库中,并获取指定 SQL 语句的结果。有 3 种类型的 Statement 对象,它们都作为在给定连接上执行 SQL 语句的包容器:Statement、PreparedStatement(从 Statement 继承而来)和 CallableStatement(从 PreparedStatement 继承而来)。它们都专用于发送特定类型的 SQL 语句:Statement 对象用于执行不带参数的简单 SQL 语句;PreparedStatement 对象提供了可以与查询信息一起预编译的一种语句类型;CallableStatement 对象是一种 PreparedStatement,它专门用于执行对数据库中的存储过程。

Statement 对象由 Connection 对象的 createStatement()方法创建,如下列代码所示:

```
Connection con=DriverManager.getConnection(url, "root","");
Statement stmt=con.createStatement();
```

Statement 接口提供了很多执行语句和获取结果的方法,常用的方法如表 11-3 所示。

表 11-3 Statement 主要方法

方　　法	功　　能
ResultSet executeQuery(String)	使用查询语句对数据库进行查询操作,用于产生单个 ResultSet 的语句,该方法可能抛出 SQLException
int executeUpdate(String)	对数据库进行新增、删除和修改操作,返回更新的行数
void close()	释放 Statement 对象中
boolean execute(String)	执行给定的 sql 语句,该语句返回是否有结果集
void addBatch(String)	将给定的 SQL 命令添加到此 Statement 对象的当前命令列表中
int[]executeBatch()	将一批命令提交给数据库来执行,如果全部命令执行成功,则返回更新计数组成的数组
void clearBatch()	清空此 Statement 对象的当前 SQL 命令列表

Statement 对象用于执行不带参数的简单 SQL 语句,它的典型使用是用 executeQuery 执行查询,用 executeUpdate 执行更新,如下代码:

```
Statement stmt=con.createStatement();
//以下代码执行从表 employee 中查询所有内容
ResultSet rs=stmt.executeQuery("select * from employee");
//以下代码执行删除表 employee 中一条记录
int changedLine=stme.executeUpdate("delete from employee where id='0005'");
```

PreparedStatement 对象用于执行带或不带 IN 参数的预编译 SQL 语句,它的常用方法见表 11-4 所示。

表 11-4 PrepareStatement 主要方法

方　　法	功　　能
ResultSet executeQuery()	使用查询语句对数据库进行带参数的查询操作,用于产生单个 ResultSet 的语句,该方法可能抛出 SQLException
int executeUpdate()	使用带参数 INSERT、DELETE 和 UPDATE 对数据库进行新增、删除和修改操作,返回更新的行数

续表

方　　法	功　　能
void setInt(int,int)	设定整数类型数值给 PreparedStatement 类对象的 IN 参数，第一个参数为参数顺序
void setFloat(int,float)	设定浮点数类型数值给 PreparedStatement 类对象的 IN 参数
void setNull(int,int)	设定 NULL 类型数值给 PreparedStatement 类对象的 IN 参数
void setString(int,String)	设定 String 类型数值给 PreparedStatement 类对象的 IN 参数
void setDate(int,Date)	设定 Date 类型数值给 PreparedStatement 类对象的 IN 参数
void setTime(int,Time)	设定 Time 类型数值给 PreparedStatement 类对象的 IN 参数

PreparedStatement 会将传入的 SQL 命令事先准备好等待使用。当有单一的 SQL 语句多次执行时，用 PreparedStatement 会比 Statement 更有效率。包含于 PreparedStatement 对象中的 SQL 语句可具有一个或多个 IN 参数。IN 参数的值在 SQL 语句创建时未被指定。相反的，该语句为每个 IN 参数保留一个问号（“?”）作为占位符。每个问号的值必须在该语句执行之前，通过适当的 setXXX（ ）方法来提供。PreparedStatement 的格式如下：

```
PreparedStatement pstmt= conn.prepareStatement("insert into employee values(?,?,?,?,?)");
pstmt.setString(1, "0009");
pstmt.setString(2, "KeWang");
pstmt.setInt(3, 45);
pstmt.setString(4, "No.456 Beijing West Road");
pstmt.setString(5, "Tianjin");
pstmt.executeUpdate();
```

CallableStatement 对象用于存储过程的执行，它是 PreparedStatement 的子类，基本方法和 PreparedStatement 一样，它的使用如下：

```
String procedureSQL= "{call procedure_xxx(?,?,?)}";
CallableStatementcstmt= conn.prepareCall(procedureSQL);
cstmt.setString(1, “china”);
cstmt.setInt(2, 100);
cstmt.setDate(1, new Date());
cstmt..executeUpdate();
```

4. ResultSet 类

在 Statement 执行 SQL 语句时，有时会返回 ResultSet 结果集对象，该对象负责存储数据库查询的结果，并提供一系列方法对数据库进行增加、删除和修改操作。这个类抽象了运行 SQL 中 select 语句的结果，提供了逐行访问记录的方法。ResultSet 的主要方法如表 11-5 所示。

表 11-5　ResultSet 主要方法

方　　法	功　　能
boolean absolute(int)	定位指针到指定的记录
void beforeFirst()	定位指针到第一条记录之前
void afterLast()	定位指针到最后一条记录之后
boolean first()	定位指针到第一条记录
boolean last()	定位指针到最后一条记录
boolean next()	向下移动一条记录
boolean previous()	向上移动一条记录
String getString(int) String getString(String)	返回数据库中 varchar 和 char 类型的数据，参数有两种，一个是给出所在列编号，另一个是给出所在列的列名
float getFloat(int) float getFloat(String)	返回数据库中 Float 类型的数据，参数有两种，一个是给出所在列编号，另一个是给出所在列的列名
double getDouble(int) double getDouble(String)	返回数据库中 Double 类型的数据，参数有两种，一个是给出所在列编号，另一个是给出所在列的列名
Date getDate(int) Date getDate(String)	返回数据库中 Date 类型的数据，参数有两种，一个是给出所在列编号，另一个是给出所在列的列名
int getInt(int) int getInt(String)	返回数据库中 int 类型的数据，参数有两种，一个是给出所在列编号，另一个是给出所在列的列名
int getRow()	返回当期的行号

方法 getXXX 提供了获取当前行中的某列值的途径，这个方法在数据库开发中经常使用。这个方法的两种参数中，列号是从左向右编号的，最左边的列号是 1，向右类推；列表名是 SQL 中 select 返回的列名称，在 getXXX 方法中忽略大小写。

11.2　数据库的访问

在前一节中介绍了 JDBC 的基本结构和 JDBC 的几种数据库连接方式，并且还介绍了数据库开发中常用的 JDBC 接口。本节中将通过一个在 MySQL 数据库上简单的数据表的操作向读者说明在 JSP 中如何实现数据的插入、查询、更新和删除等操作。

在介绍数据库数据操作前，需要先设计本节例子中用到的数据库，数据库软件选用 MySQL，假设读者都已经安装了这个数据库环境，并已经进行了初步的配置。本节的例子涉及的是一个表示某个公司的数据库 company，其中暂时只有一张表 employee 表示员工信息。

读者可以使用程序清单 11-2 所示的 SQL 脚本在 MySQL 中建立这个 company 数据库，并在数据库中加入的表格 employee。当然真正的数据库远远比这个复杂多了，这个数据库只是用来学习 JSP 的数据库基本操作，已经够用了。

程序清单 11-2：

```
/* newtable.sql */
```

```
create database if not exists company;
use company;
create table if not exists employee (
    id varchar(5),
    name varchar(20),
    age int,
    address varchar(50),
    city varchar(20),
    constraint primary key pk_employee(id)
);
```

程序清单 11-2 用 create database 建立了数据库 company，在其中用 create table 建立了表 employee，employee 表的结构如表 11-6 所示。

表 11-6　employee 表结构（员工信息表）

字段名称	类　型	宽　度	允许空值	键	说　明
id	varchar	5	NOT NULL	PK	员工编号
name	varchar	20			姓名
age	Int				年龄
address	varchar	50			地址
city	varchar	20			所在城市

11.2.1　插入记录

刚建立的 employee 表是一张空表，里面没有数据，所以程序清单 11-3 将 5 个员工的数据添加到表中。运行结果如图 11-8 所示。

程序清单 11-3：

```
<!--insert.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
<body>
<h1>添加数据到表 employee</h1>
<%
  try{
    //装载驱动程序
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    //连接字符串
    String url ="jdbc:mysql://localhost/company";
    //建立连接
    Connection conn=DriverManager.getConnection(url,"root","");
    //使用 PreparedStatement
```

```
        PreparedStatement pstmt=conn.prepareStatement("insert into employee values(?,?,?,?,?)");
        //第 1个员工
        pstmt.setString(1,"00001");
        pstmt.setString(2,"LiMing");
        pstmt.setInt(3,45);
        pstmt.setString(4,"No.77 Changgan Road");
        pstmt.setString(5,"Beijing");
        pstmt.execute();
        //第 2个员工
        pstmt.setString(1,"00002");
        pstmt.setString(2,"WangMing");
        pstmt.setInt(3,28);
        pstmt.setString(4,"No.88 Zhonghua Road");
        pstmt.setString(5,"Beijing");
        pstmt.execute();
        //第 3个员工
        pstmt.setString(1,"00003");
        pstmt.setString(2,"ZhangXiaogang");
        pstmt.setInt(3,40);
        pstmt.setString(4,"No.23 Gardon Road");
        pstmt.setString(5,"Shanghai");
        pstmt.execute();
        pstmt.close();
        //使用 Statement
        Statement stmt=conn.createStatement();
        //第 4个和第 5个员工
        stmt.execute("insert into employee values('00004','LiuLi',35,'No.23 Gardon Road','Shanghai')");
        stmt.execute("insert into employee values('00005','HongXiaoXiao',25,'No.777 Zhongshan Road','Nanjing
        ')");
        out.println("添加数据成功");
        //关闭连接、释放资源
        stmt.close();
        conn.close();
      }catch(ClassNotFoundException cnfe){
          out.print(cnfe);
      }catch(SQLException sqle){
        out.print(sqle);
      }catch(Exception e){
        out.print(e);
      }
    %>
    </body>
    </html>
```

在程序清单 11-3 中,分别使用了 PreparedStatement 和 Statement 对象向数据库 company 的表 employee 中添加员工信息。这时查询 MySQL 数据,可以看到表中已经有 5



图 11-8　添加数据

个员工的信息了,如图 11-9 所示。

```
C:\WINDOWS\system32\cmd.exe - mysql -u root
mysql> use company
Database changed
mysql> select * from employee;
+-------+---------------+------+---------------------+----------+
| id    | name          | age  | address             | city     |
+-------+---------------+------+---------------------+----------+
| 00001 | LiMing        |   45 | No.77 Changgan Road | Beijing  |
| 00002 | WangMing      |   28 | No.88 Zhonghua Road | Beijing  |
| 00003 | ZhangXiaogang |   40 | No.23 Gardon Road   | Shanghai |
| 00004 | LiuLi         |   35 | No.23 Gardon Road   | Shanghai |
| 00005 | HongXiaoXiao  |   25 | No.777 Zhongshan Road | Nanjing |
+-------+---------------+------+---------------------+----------+
5 rows in set (0.00 sec)

mysql> _
```

图 11-9　emplyee 表查询数据

11.2.2　查询记录

查询记录是数据库开发中使用最频繁的操作,它的基本顺序和更新一样,只是使用 SQL 中的 select 语句,往往在程序中还需要对查询的结果进行处理。程序清单 11-4 演示了一个对 employee 表进行查询的例子。

程序清单 11-4:

```
<!-- select.jsp -->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
<body>
<h1>查询 employee</h1>
<table border=1 cellspacing=1>
<tr><td>员工编号</td><td>姓名</td><td>年龄</td><td>地址</td><td>城市</td></tr>
<%
  try{//装载驱动程序
    Class.forName("com.mysql.jdbc.Driver").newInstance();   //连接字符串
    String url ="jdbc:mysql://localhost/company";          //建立连接
```

```
        Connection conn=DriverManager.getConnection(url,"root","");  //建立 Statement
        Statement stmt=conn.createStatement();  //执行查询建立 ResultSet
        ResultSet rs=stmt.executeQuery("select * from employee");//输出查询结果
        while(rs!=null && rs.next()){
          out.print("<tr><td>"+rs.getString("id")+"</td>");
          out.print("<td>"+rs.getString("name")+"</td>");
          out.print("<td>"+rs.getInt("age")+"</td>");
          out.print("<td>"+rs.getString("address")+"</td>");
          out.print("<td>"+rs.getString("city")+"</td></tr>");
        }
        //关闭连接、释放资源
        rs.close();
        stmt.close();
        conn.close();
      }catch(ClassNotFoundException cnfe){
          out.print(cnfe);
      }catch(SQLException sqle){
        out.print(sqle);
      }catch(Exception e){
        out.print(e);
      }
%>
</table>
</body>
</html>
```

在程序清单 11-4 中,通过 Statement 对象 stmt 的 executeQuery 方法执行 select * from employee 语句,并将查询后的结果返回到 ResultSet 对象 rs 中,最后利用 getXXX 方法在表格中输出查询的效果,运行结果如图 11-10 所示。



员工编号	姓名	年龄	地址	城市
00001	LiMing	45	No. 77 Changgan Road	Beijing
00002	WangMing	28	No. 88 Zhonghua Road	Beijing
00003	ZhangXiaogang	40	No. 23 Gardon Road	Shanghai
00004	LiuLi	35	No. 23 Gardon Road	Shanghai
00005	HongXiaoXiao	25	No. 777 Zhongshan Road	Nanjing

图 11-10　查询数据

11.2.3 更新记录

在数据库中更新数据使用 SQL 中的 update 语句，程序清单 11-5 在 employee 表中把 00001 号员工年龄改为 48 岁。

程序清单 11-5：

```
<!--update.jsp -->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
<body>
<h1>更新 employee</h1>
<%
  try{
    //装载驱动程序
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    //连接字符串
    String url ="jdbc:mysql://localhost/company";
    //建立连接
    Connection conn=DriverManager.getConnection(url,"root","");
    //建立 Statement
    Statement stmt=conn.createStatement();
    //执行 update 更新
    String sql="update employee set age=48 where id='00001'";
    int result=stmt.executeUpdate(sql);
    //输出更新结果
    out.print(result+"条记录被更新了");
    //关闭连接、释放资源
    stmt.close();
    conn.close();
  }catch(ClassNotFoundException cnfe){
      out.print(cnfe);
  }catch(SQLException sqle){
    out.print(sqle);
  }catch(Exception e){
    out.print(e);
  }
%>
</body>
</html>
```

程序清单 11-5 演示了用 executeUpdate 方法执行了 update 语句，返回更新记录的条数，在这个例子上只更新了一条记录。运行结果如图 11-11 所示。



图 11-11 更新数据

11.2.4 删除记录

删除的 SQL 语句为 delete，只要将程序清单 11-5 中的 SQL 语句修改为 delete 语句就可以实现删除操作，程序清单 11-6 中删除了员工号为 00005 的员工，运行结果如图 11-12 所示。

程序清单 11-6：

```
<!--delete.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
<body>
<h1>删除 employee 中记录</h1>
<%
  try{
    //装载驱动程序
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    //连接字符串
    String url ="jdbc: mysql: //localhost/company";
    //建立连接
    Connection conn=DriverManager.getConnection(url,"root","");
    //建立 Statement
    Statement stmt=conn.createStatement();
    //执行 update 更新
    String sql="delete form employee where id='00005'";
    int result=stmt.executeUpdate(sql);
    //输出更新结果
    out.print(result+"条记录被删除了");
    //关闭连接、释放资源
    stmt.close();
    conn.close();
  }catch(ClassNotFoundException cnfe){
      out.print(cnfe);
  }catch(SQLException sqle){
```

```
        out.print(sqle);
    }catch(Exception e){
        out.print(e);
    }
%>
</body>
</html>
```



图 11-12　删除数据

11.2.5　JSP 访问数据库的应用实例

前面介绍了数据库的基本操作，在实际开发中，远远比这个复杂，但是基本的操作步骤和上文的介绍是一样的。在这一节中间将给读者介绍一个复杂一点的数据库应用实例。

本节中的例子在上一节中的 employee 表上做了一个扩展，在 employee 中加入一个字段 departid，表示员工所属的部门编号，同时增加一个 depart 表，表示该公司的部门信息。修改的 employee 和新增 depart 表的结构如表 11-7 和 11-8 所示。

表 11-7　修改的 employee 表结构(员工信息表)

字段名称	类　　型	宽　度	允许空值	键	说　　明
id	varchar	5	NOT NULL	PK	员工编号
name	varchar	20			姓名
age	Int				年龄
address	varchar	50			地址
city	varchar	20			所在城市
departid	varchar	5		FK	所在部门编号

表 11-8　depart 表结构(部门信息表)

字段名称	类　　型	宽　度	允许空值	键	说　　明
departid	varchar	5	NOT NULL	PK	部门编号
departname	varchar	20			部门名称
dpartaddress	varchar	50			部门办公地点

读者可以执行程序清单 11-7 中的 SQL 脚本，来更新数据库 company。

程序清单 11-7：

```
/* edittable.sql */
use company;
create table if not exists depart (
     departid varchar(5),
     departname varchar(20),
     departaddress varchar(50),
     constraint primary key pk_depart(departid)
);
alter table  employee add departid varchar(5);
alter table  employee add constraint foreign key fk_employee(departid)
references depart(departid);
```

在这个例子中，模拟管理员对公司员工和部门信息进行管理，两个表之间通过外键 departid 建立联系，这种数据库操作是在实际开发中经常用到的，主要涉及用户的管理和员工的管理两个功能，整个应用的文件分为“首页”、“部门管理”和“员工管理”这 3 块。

1. 首页

首页实现两个信息管理的入口，用到的仅仅是静态的 html 标签，代码如程序清单 11-8 所示。

程序清单 11-8：

```
<!--index.html-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>数据库实例</title>
    <meta http-equiv="content-type" content="text/html;charset=gb2312">
  </head>
<body>
   <p><a href="depart.jsp">部门管理</a></p>
   <p><a href="employee.jsp">员工管理</a></p>
  </body>
</html>
```

从这个文件可以看出，部门管理是定位与 depart.jsp 文件，员工管理是定位于 employee.jsp 文件，运行结果如图 11-13 所示。



图 11-13　index.html

2. 部门管理

部门管理涉及到部门的添加，删除，修改，相关文件有以下 5 个。

depart.jsp：用于显示现有部门信息，并实现操作转发。

departform.jsp：部门信息表单，添加操作和修改共用这个

文件。

departadd.jsp：部门添加。

departedit.jsp：部门修改。

departdelete.jsp：部门删除。

（1）depart.jsp。在这个文件中，通过查询数据库将现有的部门用表格形式表示，见程序清单 11-9：

程序清单 11-9：

```
<!--depart.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
  <head>
    <title>部门管理</title>
  </head>
<%
  try{
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    String url ="jdbc:mysql://localhost/company";
    Connection conn=DriverManager.getConnection(url,"root","");
    Statement stmt=conn.createStatement();
%>
  <body>
  部门管理页面|<a href="index.html">回到首页</a>
  <hr>
  部门信息如下|<a href="departform.jsp? action=1">新建一个部门</a><br>
  <table border=1 cellspacing=1>
  <tr>
    <td>部门</td><td>部门名称</td><td>办公地点</td>
    <td>修改</td><td>删除</td>
  </tr>
<%
    String sql="select * from depart";
    ResultSet rs=stmt.executeQuery(sql);
    while(rs!=null && rs.next()){
      out.print("<tr><td>"+rs.getString("departid")+"</td>");
      out.print("<td>"+rs.getString("departname")+"</td>");
      out.print("<td>"+rs.getString("departaddress")+"</td>");
      out.print("<td><a href=\"departform.jsp? action=2&departid="+
          rs.getString("departid")+"\">修改</a></td>");
      out.print("<td><a href=\"departdelete.jsp?departid="+
          rs.getString("departid")+"\">删除</a></td></tr>");
    }
%>
  </table>
```

```
    </body>
  <%
      rs.close();stmt.close(); conn.close();
    }catch(ClassNotFoundException cnfe){
        out.print(cnfe);
    }catch(SQLException sqle){
      out.print(sqle);
    }catch(Exception e){
      out.print(e);
    }
  %>
  </html>
```

在这个文件中，对部门实现添加、修改和删除都是使用超链接的方式实现，在修改和删除超链接中通过加入部门编号定位需要修改和删除的部门记录。结果如图 11-14 所示。



图 11-14　显示部门

（2）departform.jsp。为了方便添加和修改，在文件中设计了一个表单，用文本框表示部门的信息，同时使用这个表单也可以提交修改和新建的信息。添加和修改共用了这个文件，通过 action 参数来区分这两个操作，departform.jsp? action=1 表示添加操作，departform.jsp? action=2 表示修改操作，文件如程序清单 11-10 所示。

程序清单 11-10：

```
  <!--departform.jsp-->
  <%@page contentType="text/html; charset=gb2312" language="java"%>
  <%@page import="java.sql.*"%>
  <html>
    <head><title>部门管理</title></head>
  <%
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    String url ="jdbc: mysql: //localhost/company";
    Connection conn=DriverManager.getConnection(url,"root","");
    Statement stmt=conn.createStatement();
```

```
    ResultSet rs= stmt.executeQuery("select * from depart");
    String action= request.getParameter("action");
    String actiontype= "";
    String actionaim= "";
    String editdepartid= "";
    if(action.equals("1")){
      actiontype= "新建一个部门";
      actionaim= "departadd.jsp";
    }else{
      actiontype= "修改部门信息";
      actionaim= "departedit.jsp";
      editdepartid= request.getParameter("departid");
      rs= stmt.executeQuery("select * from depart where departid= '"+ editdepartid+ "'");
      if(rs!= null)rs.next();
    }
  %>
    <body>
      <%= actiontype%> | <a href= "index.html">回到首页</a><hr>
     <form action= <%= actionaim%> method= post>
       <table border= 0 cellspacing= 0>
       <tr><td>  部门编号:</td>
         <td><%
           if(action.equals("1"))
             out.print("<input type= \"text\" size= \"10\" name= \"departid\">");
           else
             out.print("<input type= \"text\" size= \"10\" name= \"departid\" readonly= true
               value= "+ rs.getString("departid")+ ">");
         %></td></tr>
       <tr><td>  部门名称:</td>
         <td><%
           if(action.equals("1"))
             out.print("<input type= \"text\" size= \"20\" name= \"departname\">");
           else
             out.print("<input type= \"text\" size= \"20\" name= \"departname\"
                 value= \""+ rs.getString("departname")+ "\" >");
         %></td></tr>
       <tr><td>  办公地点:</td>
         <td><%
           if(action.equals("1"))
             out.print("<input type= \"text\" size= \"40\" name= \"departaddress\">");
           else
             out.print("<input type= \"text\" size= \"40\" name= \"departaddress\"
                     value= \""+ rs.getString(3)+ "\" >");
         %></td></tr>
       </table>
```

```
        <input type=submit value="提交">
      </form>
    </body>
<% rs.close(); stmt.close(); conn.close();%>
</html>
```

运行结果如图 11-15 所示。

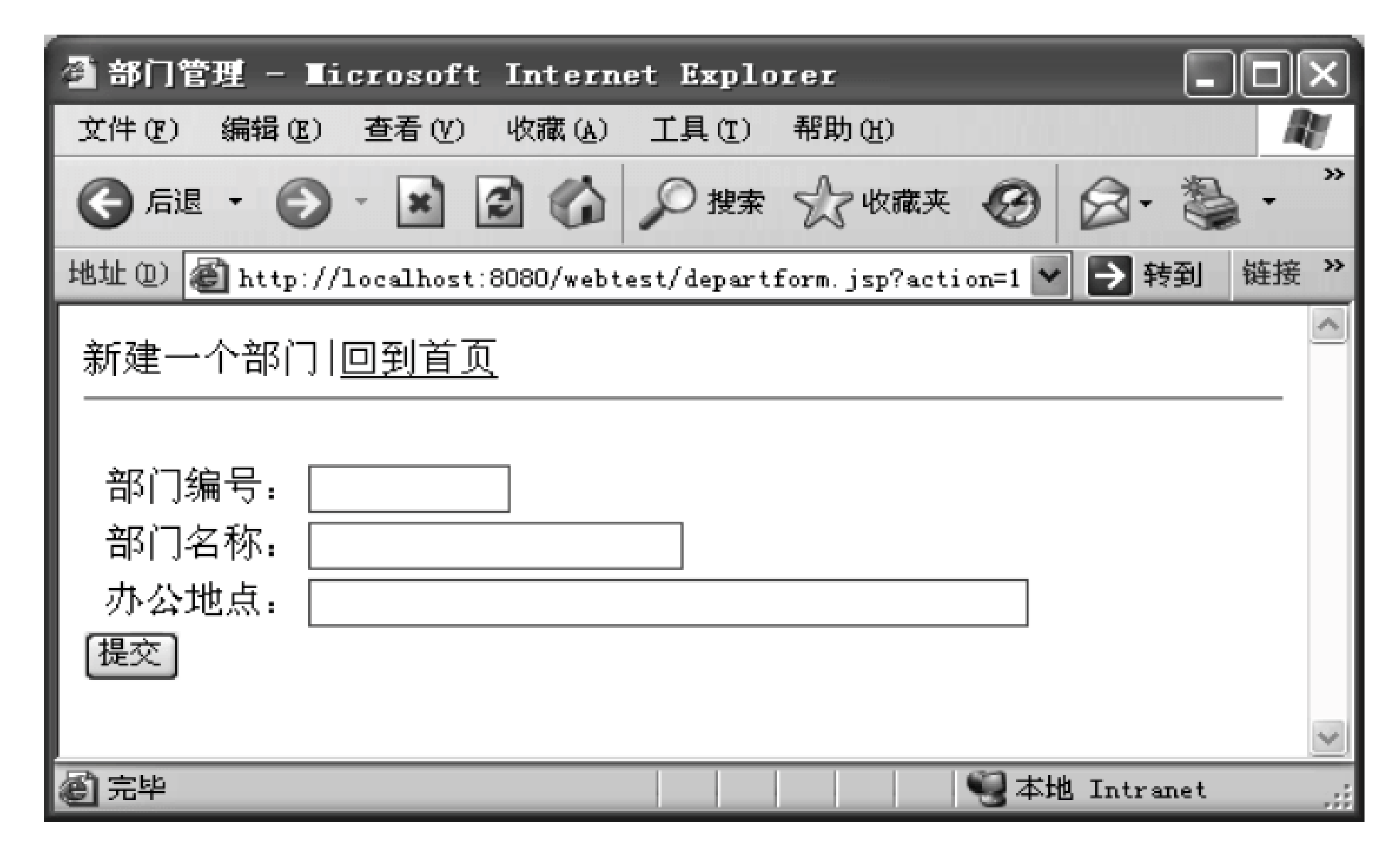


图 11-15　新建部门

(3) departadd.jsp。如果给出的参数是 action=1,则表单执行的是添加操作,在表单中填入数据后,然后提交到 departadd.jsp 实现对数据库的插入,见程序清单 11-11。

程序清单 11-11:

```
<!--departadd.jsp-->
<%@ page contentType="text/html; charset=gb2312" language="java"%>
<%@ page import="java.sql.*"%>
<html>
<body>
<%
  try{
    String departid=request.getParameter("departid");
    String departname=request.getParameter("departname");
    String departaddress=request.getParameter("departaddress");
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    String url="jdbc:mysql://localhost/company";
    Connection conn=DriverManager.getConnection(url,"root","");
    PreparedStatement pstmt=conn.prepareStatement("insert into depart values(?,?,?)");
    pstmt.setString(1,departid);
    pstmt.setString(2,departname);
    pstmt.setString(3,departaddress);
    pstmt.execute();
    pstmt.close();
    conn.close();
```

```
        response.sendRedirect("depart.jsp");
    }catch(Exception e){
        out.print(e);
    }
%>
</body>
</html>
```

(4) departedit.jsp。如果给出的参数是 action=2,则表单执行是的修改操作,提交后由 departedit.jsp 实现修改,见程序清单 11-12。

程序清单 11-12:

```
<!--departedit.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
<body>
<%
    try{
        String departid=request.getParameter("departid");
        String departname=request.getParameter("departname");
        String departaddress=request.getParameter("departaddress");
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        String url="jdbc:mysql://localhost/company";
        Connection conn=DriverManager.getConnection(url,"root","");
        Statement stmt=conn.createStatement();
        String sql="update depart set departname='"+
                departname+"',departaddress='"+
                departaddress+"' where departid='"+departid+"'";
out.print(sql);
        int result=stmt.executeUpdate(sql);
        stmt.close();
        conn.close();
        response.sendRedirect("depart.jsp");
    }catch(ClassNotFoundException cnfe){
          out.print(cnfe);
    }catch(SQLException sqle){
        out.print(sqle);
    }catch(Exception e){
        out.print(e);
    }
%>
</body>
</html>
```

(5) departdelete.jsp。在 departform.jsp 中通过将需要删除的部门编号 departid 传送

给 departdelete.jsp 文件后，实现对所选的记录的删除。如程序清单 11-13。

程序清单 11-13：

```
<!--departdelete.jsp -->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
<body>
<%
  try{
    String departid=request.getParameter("departid");
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    String url="jdbc:mysql://localhost/company";
    Connection conn=DriverManager.getConnection(url,"root","");
    Statement stmt=conn.createStatement();
    String sql="delete from depart where departid='"+departid+"'";
    int result=stmt.executeUpdate(sql);
    stmt.close(); conn.close();response.sendRedirect("depart.jsp");
  }catch(Exception e){
    out.print(e);
  }
%>
</body>
</html>
```

3. 员工管理

更重要的操作是对员工资料的管理，即实现员工资料的查询添加，修改，删除方法。员工表 employee 是通过 departid 外键访问到表 depart 来得到部门名称这个信息，这就需要关联到两个表格，而且员工的所在部门可选范围一定是局限在表 depart 中已有的数据，这样就增加了数据库操作的复杂度，但这是读者将来在开发 JSP 程序中经常会遇到的结构，所以在下文中对这类结构的处理方法做一个介绍。相关文件有以下 5 个。

employee.jsp：用于显示现有员工信息，并实现操作转发。

employeeform.jsp：员工信息表单，添加操作和修改共用这个文件。

employeeadd.jsp：员工添加。

employeeedit.jsp：员工修改。

employeedelete.jsp：员工删除。

（1）employee.jsp。这个文件用于显示员工的资料，由于部门的信息存放在 depart 表中，所以这里的查询需要考虑到两个表的情况，由于考虑到员工有可能还没有设定部门，所以两个表的连接采用了自然左外连接的方式 select * from (employee natural left outer join depart)，见程序清单 11-14，运行结果见图 11-16。

程序清单 11-14

```
<!--employee.jsp -->
<%@page contentType="text/html; charset=gb2312" language="java"%>
```

```
<%@page import="java.sql.*"%>
<html>
  <head><title>员工管理</title></head>
<%
  try{
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    String url="jdbc:mysql://localhost/company";
    Connection conn=DriverManager.getConnection(url,"root","");
    Statement stmt=conn.createStatement();
%>
  <body>
  员工管理页面|<a href="index.html">回到首页</a>
  <hr>
  员工信息如下|<a href="employeeform.jsp?action=1">新建一个员工</a><br>
  <table border=1 cellspacing=1>
  <tr>
    <td>员工编号</td><td>姓名</td>
      <td>年龄</td><td>地址</td><td>城市</td><td>所属部门</td>
    <td>修改</td><td>删除</td>
  </tr>
<%
    String sql="select * from (employee natural left outer join depart)";
    ResultSet rs=stmt.executeQuery(sql);
    while(rs!=null && rs.next()){
      out.print("<tr><td>"+rs.getString("id")+"</td>");
      out.print("<td>"+rs.getString("name")+"</td>");
      out.print("<td>"+rs.getInt("age")+"</td>");
      out.print("<td>"+rs.getString("address")+"</td>");
      out.print("<td>"+rs.getString("city")+"</td>");
      out.print("<td>"+rs.getString("departname")+"</td>");
      out.print("<td><a href=\"employeeform.jsp?action=2&id="+
          rs.getString("id")+"\">修改</a></td>");
      out.print("<td><a href=\"employeedelete.jsp?id="+
          rs.getString("id")+"\">删除</a></td></tr>");
    }
%>
  </table>
  </body>
<%
    rs.close();stmt.close();conn.close();
  }catch(ClassNotFoundException cnfe){
      out.print(cnfe);
  }catch(SQLException sqle){
    out.print(sqle);
  }catch(Exception e){
```

```
        out.print(e);
    }
%>
</html>
```



图 11-16　员工列表

(2) employeeform.jsp。在这个文件中使用表单文本框表示一个员工的信息，和部门表示中一样，这个文件同时支持添加和修改，这两个操作还是通过参数 action 等于 1 或者等于 2 来区分。无论员工的添加还是修改，所能选择的部门名称只能局限在表 depart 中，所以部门信息在表单中采用了选择框的方式，选择框的生成采用一次读数据库的结果集 ResultSet 来完成，这样就可以防止用户输入非法的数据。读者在以后相关的设计中，性别、时间等信息的输入往往可以采用事先生成选择框的方式，文件见程序清单 11-15，运行结果如图 11-17 所示。

程序清单 11-15：

```
<!--employeeform.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
  <head><title>员工管理</title></head>
<%
  Class.forName("com.mysql.jdbc.Driver").newInstance();
  String url="jdbc:mysql://localhost/company";
  Connection conn=DriverManager.getConnection(url,"root","");
  Statement stmt=conn.createStatement();
  ResultSet rs=stmt.executeQuery("select * from employee");
  String action=request.getParameter("action");
  String actiontype="";
  String actionaim="";
  String editemployeeid="";
  if(action.equals("1")){
```

```
        actiontype="新建一个员工";
        actionaim="employeeadd.jsp";
    }else{
        actiontype="修改员工信息";
        actionaim="employeeedit.jsp";
        editemployeeid=request.getParameter("id");
         rs=stmt.executeQuery("select * from (employee natural left outer join depart) where id='"+
editemployeeid+"'");
        if(rs!=null)rs.next();
    }
 %>
   <body>
        <%=actiontype%>|<a href="index.html">回到首页</a>
     <hr>
     <form action=<%=actionaim%>method=post>
       <table border=0 cellspacing=0>
       <tr><td> 员工编号:</td>
          <td><%
            if(action.equals("1"))
              out.print("<input type=\"text\" size=\"10\" name=\"id\">");
            else
              out.print("<input type=\"text\" size=\"10\" name=\"id\" readonly=true
                  value="+rs.getString("id")+">");
          %></td></tr>
          //…
          //此处省略姓名,年龄,地址,城市输入框,格式和上面员工编号类似
          //…
          <tr><td> 所属部门:</td><td>
          <select name="departid">
          <%
            String str="";
            if(action.equals("2"))str=rs.getString(7);
            ResultSet temprs=stmt.executeQuery("select * from depart");
            while(temprs!=null && temprs.next()){
              out.print("<option value=\""+temprs.getString("departid")+"\"");
              if(action.equals("2")&& str.equals(temprs.getString("departname")))
                out.print("selected");
              out.print(">"+temprs.getString("departname")+"</option>");
            }
            temprs.close();
          %>
          </select></td></tr>
          </table>
       <input type=submit value="提交">
     </form>
```

```
  </body>
<%
  rs.close(); stmt.close(); conn.close();%>
</html>
```

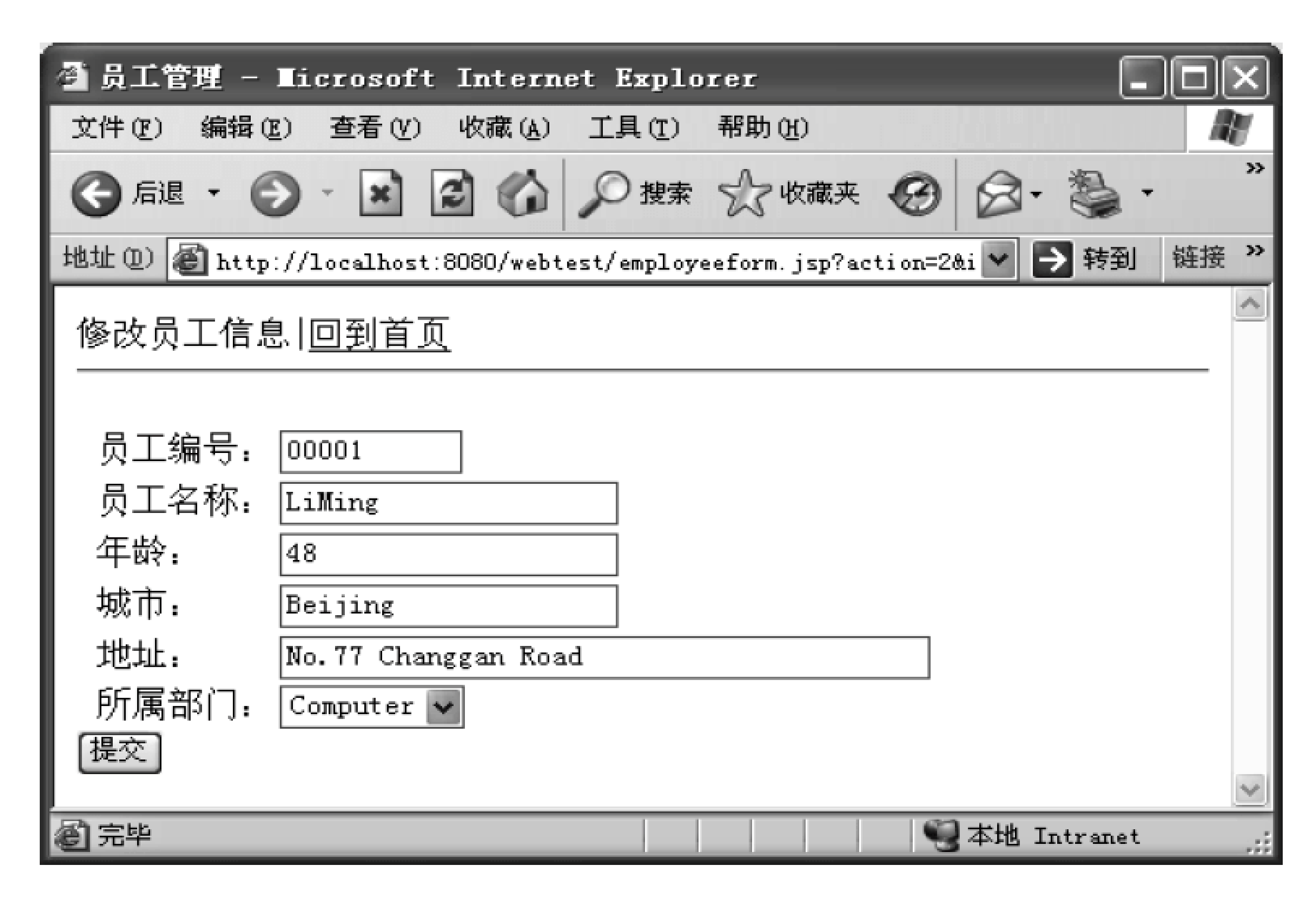


图 11-17　修改员工

(3) employeeadd.jsp。当 employeeform.jsp 是表示添加员工的时候,由 employeeadd.jsp 实现了员工的数据添加,见程序清单 11-16。

程序清单 11-16:

```
<!--employeeadd.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
<body>
<%
  try{
    String id=request.getParameter("id");
    String name=request.getParameter("name");
    int age=Integer.valueOf(request.getParameter("age")).intValue();
    String address=request.getParameter("address");
    String city=request.getParameter("city");
    String departid=request.getParameter("departid");
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    String url="jdbc:mysql://localhost/company";
    Connection conn=DriverManager.getConnection(url,"root","");
    PreparedStatement pstmt=conn.prepareStatement("
                          insert into employee values(?,?,?,?,?,?)");
    pstmt.setString(1,id);
    pstmt.setString(2,name);
```

```
        pstmt.setInt(3,age);
        pstmt.setString(4,address);
        pstmt.setString(5,city);
        pstmt.setString(6,departid);
        pstmt.execute();pstmt.close(); conn.close();
        response.sendRedirect("employee.jsp");
    }catch(Exception e){
        out.print(e);
    }
%>
</body>
</html>
```

(4) employeeedit.jsp。当 employeeform.jsp 是表示修改员工信息的时候，由 employeeedit.jsp 实现了员工的修改，见程序清单 11-17。

程序清单 11-17：

```
<!--employeeedit.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
<body>
<%
    try{
        String id=request.getParameter("id");
        String name=request.getParameter("name");
        int age=Integer.valueOf(request.getParameter("age")).intValue();
        String address=request.getParameter("address");
        String city=request.getParameter("city");
        String departid=request.getParameter("departid");
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        String url="jdbc: mysql: //localhost/company";
        Connection conn=DriverManager.getConnection(url,"root","");
        Statement stmt=conn.createStatement();
        String sql="update employee set name='"+
                name+"',age="+
                age+",address='"+
                address+"',city='"+
                city+"',departid='"+
                departid+"' where id='"+id+"'";
        int result=stmt.executeUpdate(sql);
        stmt.close(); conn.close(); response.sendRedirect("employee.jsp");
    }catch(Exception e){
        out.print(e);
    }
%>
```

```
</body>
</html>
```

（5）employeedelete.jsp。employeedelete.jsp 这个文件实现了删除记录。见程序清单 11-18。

程序清单 11-18：

```
<!--employeedelete.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html>
<body>
<%
  try{
    String id=request.getParameter("id");
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    String url="jdbc:mysql://localhost/company";
    Connection conn=DriverManager.getConnection(url,"root","");
    Statement stmt=conn.createStatement();
    String sql="delete from employee where id='"+id+"'";
    int result=stmt.executeUpdate(sql);
    stmt.close(); conn.close();response.sendRedirect("employee.jsp");
  }catch(Exception e){
    out.print(e);
  }
%>
</body>
</html>
```

11.3　数据库访问常用技巧

11.3.1　中文字符乱码问题的解决

在 JSP 开发中，往往由于编码方式的不一致，导致中文字符出现乱码的情况，实际上中文乱码问题不仅仅出现在数据库开发中，凡是用到中文字符读取、存储、显示的地方都有可能遇到乱码问题。其中有 3 处是读者有可能经常会遇到的，第一是页面显示中文乱码，第二是使用 request 读取表单内容的时候，第三就是涉及数据库存取的时候。在本节中将对中文乱码问题做一个简要的说明，并给出解决方法。

产生这个问题的原因说起来也很简单，由于 JSP 中文编码和默认的 Web 服务器，浏览器的编码不一致造成的。根据现在的编码特点字符编码基本可以分为以下 3 种。

（1）单字节字符编码 ISO-8859-1。最简单的编码规则，每一个字节直接转化作为一个 UNICODE(2 个字节)字符。将 UNICODE 字符串反向通过 ISO-8859-1 转化为字节串时，只能正常转化 0～255 范围的字符，就是早期的 ASCII 编码规定的那些字符。

(2) ANSI 编码。ISO-8859-1 编码只能表示少量字符，所以不同的国家和地区制定了不同的标准，由此产生了 GB 2312、BIG5、JIS 等本地性的编码标准。这些使用 2 个字节来代表一个字符的各种字符延伸编码方式，称为 ANSI 编码。在简体中文操作系统下，ANSI 编码代表 GB 2312 编码，在日文操作系统下，ANSI 编码代表 JIS 编码。不同 ANSI 编码之间互不兼容，当信息在国际间交流时，无法将属于两种语言的文字，存储在同一段 ANSI 编码的文本中。如果字符是以 ANSI 编码形式存在的，一个字符可能使用一个字节或多个字节来表示，那么我们称这种字符串为 ANSI 字符串或者多字节字符串

(3) UNICODE 编码。为了使国际间信息交流更加方便，国际组织制定了 UNICODE 字符集，为各种语言中的每一个字符设定了统一并且唯一的数字编号，以满足跨语言、跨平台进行文本转换、处理的要求，UNICODE 编码和 ISO-8859-1 编码可以对应转换。在内存中，如果“字符”是以在 UNICODE 中的序号存在的，那么我们称这种字符串为 UNICODE 字符串或者宽字节字符串。

在 JSP 开发中，由于中文操作系统默认是使用 GB 2312 编码，Tomcat 等 JSP 服务器和部分数据库软件使用 ISO-8859-1 编码，而 Java 代码又默认使用 UNICODE 编码，所以产生了中文字符的乱码问题。解决这个问题，只需要将各处的编码方式统一即可。

对于上文提到的 3 个常见的中文乱码现象可以用下列方式分别解决。

(1) 页面显示中文乱码。只要在 Web 页码加入如下代码，手工指定页面的编码方式就可以解决：

```
<%@page contentType="text/html; charset=gb2312">
```

(2) 使用 request 读取表单内容的乱码。由于 JSP 服务器 Tomcat 默认采用 ISO-8859-1 编码执行 request 和 response。所以只需要在用 request 读取表单属性的时候，用下来语句修改 request 方法的编码方式就可以解决：

```
request.setCharacterEncoding("gb2312");
```

当然这句话要放在执行 request.getParameter()方法之前。

(3) 数据库存取中的乱码。数据库的乱码也是由于数据库软件是采用 ISO-8859-1 编码对应的 UNICODE 编码方式存储数据内容，而中文字符从数据库中读出后，要在相应的操作系统上显示(比如在简体中文的操作中，采用 GB 2312 编码)，就会出现乱码。解决的方式是将 UNICODE 字符串转换为 GB 2312 的字符串。

程序清单 11-19 是在对程序清单 11-10 进行的修改，修改后代码将从数据库读取出的字符转换为 GB 2312 编码方式字符串，这样就解决了显示乱码问题。

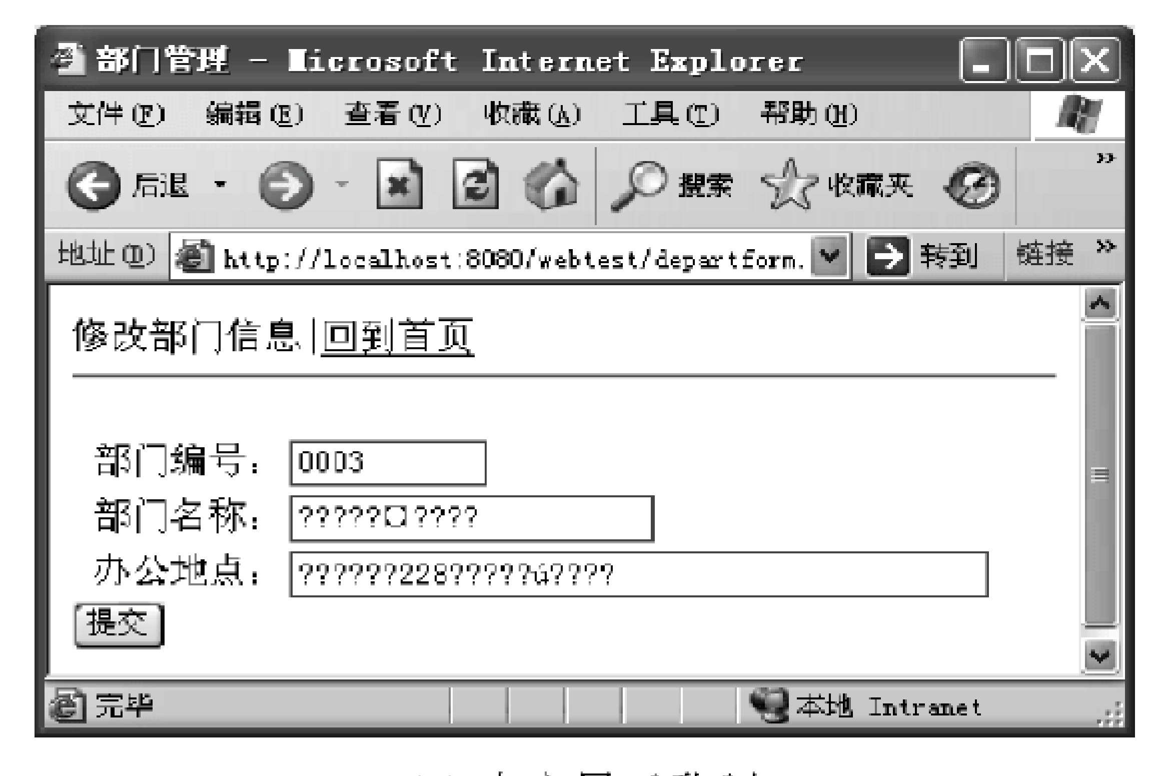
程序清单 11-19：

```
<!--departform.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*,java.io.*"%>
<html><head><title>部门管理</title></head>
<%!
//定义了一个字符串转换函数 trans
String trans(String origin){
```

```
    String result=null;
    try{
      byte[] temp=origin.getBytes("iso-8859-1");
      result=new String(temp,"gb2312");
    }catch(UnsupportedEncodingException usee){}
    return result;
  }
  %>
  //此处省略程序清单 11-10 中一样的代码,读者可参考程序清单 11-10
  <body>
      <%=actiontype%>|<a href="index.html">回到首页</a>
    <hr>
    <form action=<%=actionaim%>method="POST">
      <table border=0 cellspacing=0>
      <tr>
        <td> 部门编号:</td>
        <td>
        <%
          if(action.equals("1"))
            out.print("<input type=\"text\" size=\"10\" name=\"departid\">");
          else
            //对 rs.getString("departid")返回的字符串转换编码格式
            out.print("<input type=\"text\" size=\"10\" name=\"departid\" readonly=true
                value="+trans(rs.getString("departid"))+">");
        %>
        </td></tr>
  //代码省略…
            out.print("<input type=\"text\" size=\"20\" name=\"departname\"
                value=\""+trans(rs.getString("departname"))+"\">");
  //代码省略…
            out.print("<input type=\"text\" size=\"40\" name=\"departaddress\"
                value=\""+trans(rs.getString("departaddress"))+"\">");
  //代码省略…
```

在程序清单 11-19 中加入一个转换方法 trans(),将读出的字符串进行转换。运行效果如图 11-18 所示,将图 11-18(a)中乱码成功修正为图 11-18(b)中正常显示。

程序清单 11-19 中实现了对数据库读取后字符的转换,当然,采用这种方式,写入数据库之前也要将 GB 2312 编码的字符串转换为 UNICODE 编码的字符串再写入数据表中。读者可能会觉得这样非常麻烦,实际上也有“一劳永逸”的方法,部分数据库软件可以直接定义数据库存储的字符编码格式,只要读者在开发的时候,将数据库修改为采用 GB 2312 方式存储就可以免去在读写中对字符串进行转换。比如在 MySQL 数据库中,读者只需要将配置文件 my.ini 中默认的 default-character-set=latin1 属性修改为 default-character-set=gb2312 就可以实现以 GB 2312 编码方式存数据。



(a) 中文显示乱码



(b) 修正乱码问题

图 11-18　数据库读取乱码处理

11.3.2　分页显示的方法

在数据库开发中，查询是频繁使用的操作，往往需要将查询得到的数据在 Web 页面中显示出来，当查询的结果非常多的时候，只用一页显示全部记录就不是太合适，这就需要用到分页显示的方法。分页显示也是一个在数据库开发中经常需要用到技术，希望读者能够通过本节的学习掌握这个技术。

总体来说，分页显示有两种方法，第一种是先查询出所有记录放入结果集，但一次只从结果集中取出一部分需要的记录；另一种是只查询出需要的内容放入记录集。

先介绍第一种方法，这种方法是传统的方法，首先将查询得到的全部结果放入 ResultSet，然后根据当前需要读的页数计算出从第几行开始读取，需要读取多少个，再将 ResultSet 定位到需要读取的位置。程序清单 11-20 演示了对 employee 表中数据的分页显示。结果见图 11-19 所示。

程序清单 11-20：

```
<!--employeefy1.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html><head><title>员工管理</title></head>
<%
  int intPageSize=4;            //一页显示的记录数,此处为了演示,初始化为 4
  int intRowCount=0;            //记录总数
  int intPageCount=0;           //总页数
  int intPage;                  //需要显示页码
  String strPage;               //待显示页码字符串格式
  int i;
  try{
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    String url="jdbc: mysql: //localhost/company";
    Connection conn=DriverManager.getConnection(url,"root","");
    Statement stmt=conn.createStatement();
```

```
        String sql="select * from (employee natural left outer join depart)";
        ResultSet rs=stmt.executeQuery(sql);
        if(rs!=null){
          rs.last();
          intRowCount=rs.getRow();        //获得总记录数
          intPageCount=(intRowCount+intPageSize-1)/intPageSize;   //获得页数
        }
        strPage=request.getParameter("page");
        if(strPage==null)
          intPage=1;
        else{
          intPage=Integer.valueOf(strPage).intValue();
          if(intPage<1)intPage=1;
          if(intPage>intPageCount)intPage=intPageCount;
        }
        if(intPageCount>0)
          rs.absolute((intPage-1)*intPageSize+1);
%>
    <body>
    员工管理页面|<a href="index.html">回到首页</a>
    <hr>
    员工信息如下|<a href="employeeform.jsp?action=1">新建一个员工</a><br>
    <table border=1 cellspacing=1>
    <tr><td>员工编号</td><td>姓名</td><td>年龄</td><td>地址</td><td>城市</td>
      <td>所属部门</td><td>修改</td><td>删除</td></tr>
<%
        i=0;
        while(i<intPageSize && rs!=null && !rs.isAfterLast()){
          out.print("<tr><td>"+rs.getString("id")+"</td>");
          out.print("<td>"+rs.getString("name")+"</td>");
          out.print("<td>"+rs.getInt("age")+"</td>");
          out.print("<td>"+rs.getString("address")+"</td>");
          out.print("<td>"+rs.getString("city")+"</td>");
          out.print("<td>"+rs.getString("departname")+"</td>");
          out.print("<td><a href=\"employeeform.jsp?action=2&id="+
              rs.getString("id")+"\">修改</a></td>");
          out.print("<td><a href=\"employeedelete.jsp?id="+
              rs.getString("id")+"\">删除</a></td></tr>");
          i++;
          rs.next();
        }
%>
    </table>
    <form action="employeefy1.jsp" method="get">
<%
```

```
        if(intPage>1){
            out.print("<a href=\"employeefy1.jsp?page=1\">[首页]</a>");
            out.print("<a href=\"employeefy1.jsp?page="+(intPage-1)+"\">[前一页]</a>");
        }
        if(intPage<intPageCount){
            out.print("<a href=\"employeefy1.jsp?page="+(intPage+1)+"\">[后一页]</a>");
            out.print("<a href=\"employeefy1.jsp?page="+intPageCount+"\">[末页]</a>");
        }
        out.print("转到第:<input type=\"text\" name=\"page\" size=2>页 ");
        out.print("<input type=\"submit\" value=\"go\">");
        out.print("第"+intPage+"页,共"+intPageCount+"页 ");
    %>
        </form>
        </body>
    <%
            rs.close();
            stmt.close();
            conn.close();
        }catch(Exception e){
            out.print(e);
        }
    %>
    </html>
```



图 11-19　分页显示

第一种方法可以非常简单地实现分页显示，但因为它是一次性读取所有的记录，但实际只需要显示其中一小部分记录，而大部分记录是完全不需要的，这就大大加重了数据库连接和内存的无用负担，特别是当遇到数据非常庞大的数据库的时候，这个缺陷愈加明显。所以，在本节中介绍第二种方法，这个方法不采用全部读取所有记录的方法，而是根据每页显示记录数和需要显示的页码计算出需要读取的记录位置和记录数，这样可以避免占用大量

服务器内存。

分析程序清单 11-20，可以发现若这次需要读取 intPage 页的时候，只需要将所有查询结果中的第(intPage-1) * intPageSize 行记录开始读取，读 intPageSize 行即可。这样就可以在写查询 SQL 语句的时候加入上述计算，特别时候现在有些数据库本身定义了分段查询的 SQL 语句，使得分页更加简单。比如在 MySQL 中支持如下的 SQL select 语句：

```
SELECT * FROM table LIMIT # OFFSET #
```

其中第一个参数表示返回记录行的最大数目，第二个参数表示返回记录行的偏移量。如：

```
SELECT * FROM table LIMIT 10 OFFSET 5
```

表示返回第 6 行到第 15 行。利用这个技术可以实现第二种分页显示的方法，见程序清单 11-21。但是应该注意，在 MySQL 中支持上述语句，并不说明所有数据库软件都支持，若不支持，也可以用组合的 SQL 语句来完成同样的功能。

程序清单 11-21：

```
<!--employeefy2.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="java.sql.*"%>
<html><head><title>员工管理</title></head>
<%
  int intPageSize=4;               //一页显示的记录数,此处初始化为 4
  int intRowCount=0;               //记录总数
  int intPageCount=0;              //总页数
  int intPage;                     //待显示页码
  String strPage;                  //待显示页码字符串
  try{
    Class.forName("com.mysql.jdbc").newInstance();
    String url="jdbc:mysql://localhost/company";
    Connection conn=DriverManager.getConnection(url,"root","");
    Statement stmt=conn.createStatement();
    ResultSet temprs=stmt.executeQuery("select count(*) from employee");
    if(temprs!=null){
      temprs.next();
      intRowCount=temprs.getInt(1);    //获得总记录数
      intPageCount=(intRowCount+intPageSize-1)/intPageSize;    //获得页数
    }
    temprs.close();
    strPage=request.getParameter("page");
    if(strPage==null)
      intPage=1;
    else{
      intPage=Integer.valueOf(strPage).intValue();
      if(intPage<1)intPage=1;
      if(intPage>intPageCount)intPage=intPageCount;
```

```
        }
        String sql="select * from (employee natural left outer join depart) limit ";
        sql+=intPageSize+" offset "+(intPage-1)*intPageSize;
        ResultSet rs=stmt.executeQuery(sql);
%>
    <body>
    员工管理页面|<a href="index.html">回到首页</a>
    <hr>
    员工信息如下|<a href="employeeform.jsp?action=1">新建一个员工</a><br>
    <table border=1 cellspacing=1>
    <tr><td>员工编号</td><td>姓名</td><td>年龄</td><td>地址</td><td>城市</td>
        <td>所属部门</td><td>修改</td><td>删除</td>
    </tr>
<%
        while(rs!=null && rs.next()){
            out.print("<tr><td>"+rs.getString("id")+"</td>");
            out.print("<td>"+rs.getString("name")+"</td>");
            out.print("<td>"+rs.getInt("age")+"</td>");
//余下代码省略,和程序清单11-10一样
```

11.3.3 连接池的使用

在前面的内容中已经介绍了,如果使用JDBC直接访问数据库中的数据,每一次数据访问请求都必须经历建立数据库连接、打开数据库、存取数据和关闭数据库连接等步骤。这个方法虽然简单,但是有一定的缺陷。第一,每次读写数据库,都需要建立一次连接,而连接并打开数据库是一件既消耗资源又费时的工作;第二,由于这种传统的方式没有对数据库的连接数量进行控制,因此可能出现超过数据库处理能力的连接数量和处理请求,如果频繁发生这种数据库操作,系统的性能必然会急剧下降,甚至会导致系统崩溃。连接池技术是解决这个问题最常用的方法。

数据库连接池的基本思想是为数据库建立一个连接存储池。在程序开始运行时候,一次性建立一定数量的连接,当程序中需要连接数据库时,就从连接存储池中取出一个连接,操作完成后,将连接还给连接存储池,这样就不需要每次读写数据库时,都要装载驱动,建立连接,可以提高数据库的读写性能,特别是对于读写非常频繁的数据库程序,可以很大程度地提高数据库处理速度。而且由于连接存储池中连接的数量是确定的,就可以限制数据库的最大连接数,保证数据库服务器能够正常运行。

本节中,就以Tomcat服务器为例,给读者介绍数据库连接池的设置和使用方法。数据库连接池的设置分为两个步骤。

(1) 下载相应的数据库JDBC驱动程序并安装正确。以MySQL数据库为例,就是要确保名称为mysql-connector-java-x.x.x-bin.jar(x.x.x是MySQL数据库的版本号)的文件已经被复制到Web应用目录的/WEB-IFN/lib或者Tomcat安装目录的/lib/中。

(2) 配置Tomcat中的context标签。在需要使用数据库连接池的Web应用的配置文件<context></context>标签中,加入如下参数:

```
<Resource
    name="jdbc/jspbook"
    auth="Container"
    type="javax.sql.DataSource"
    maxActive="100"
    maxIdle="30"
    maxWait="10000"
    username="dbuser" password="dbuser" driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/company?autoReconnect=true"/>
```

其中字段的属性说明如下：

name="jdbc/jspbook"：定义连接池的名称，在程序用 jdbc/jspbook 申请连接；

maxActive="100"：最大活跃连接数，这里取值为 100，表示同时最多有 100 个数据库连接，设置为 0 表示无限制；

maxIdle="30"：最大的空闲连接数，这里取值为 30，表示即使没有数据库连接时依然可以保持 30 个空闲的连接，而不被清除，随时处于待命状态，设置为 0 表示无限制；

maxWait="10000"：最大建立连接等待时间。如果超过此时间将发生异常，这里设置为 10000，表示 10 秒后超时，设置为－1 表示无限制；

username="dbuser"：用户名

password="dbuser"：密码

driverClassName="com. mysql. jdbc. Driver"：数据库驱动程序，这里是 MySQL 驱动；

url="jdbc：mysql：//localhost：3306/company?autoReconnect=true"：连接字符串；

配置 Tomcat 的 server. xml 后，然后修改对应 Web 应用对应的 web. xml 文件，用 resource－ref 元素设置引用数据库连接池，上述的 Web 应用修改 web. xml 如程序清单 11-22 所示：

程序清单 11-22：

```
<!--web.xml-->
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

  <resource-ref>
      <description>jspbook connection</description>
      <res-ref-name>jdbc/jspbook</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
  </resource-ref>
</web-app>
```

配置完成后，就可以在 JSP 中通过向连接池申请连接的方式访问数据库了。程序清单 11-23演示了用连接池的方式连接并访问数据库的例子。

程序清单 11-23：

```
<!--employeeljc.jsp-->
<%@page contentType="text/html;charset=gb2312"%>
<%@page import="java.sql.*"%>
<%@page import="javax.sql.*"%>
<%@page import="javax.naming.*"%>
<html>
<head>
<title>数据库连接测试</title>
</head>
<body>
<%
try{
  Context initContext=new InitialContext();
  Context envContext=(Context)initContext.lookup("java:/comp/env");
  //获得在 conext 中定义的连接池对象
  DataSource ds=(DataSource)envContext.lookup("jdbc/jspbook");
  //从连接池中获得数据库连接
  Connection conn=ds.getConnection();
  //数据库连接测试
  if(conn!=null)
    out.print("<h3>数据库连接成功,恭喜你</h3>") ;
  Statement stmt=conn.createStatement();
  String sql="select user(),now()";
  ResultSet rs=stmt.executeQuery(sql);
  while(rs.next()){
    out.print("当期数据库用户为:"+rs.getString(1)+"<br>");
    out.print("当前时间为:"+rs.getString(2)+"<br>");
  }
  rs.close();
  stmt.close();
  conn.close();                       //返还数据库连接
}catch(Exception e){
  out.print(e);
}
%>
</body>
</html>
```

程序清单 11-22 中,通过 ds.getConnection()申请连接,通过 conn.close()关闭连接,当然这里的关闭并不是指物理连接的关闭,而是将连接返回给连接池。

小　　结

本章主要介绍了在 JSP 中如何对数据库编程,在 Java 系统中提供了 JDBC 接口技术实现对数据库的连接和操作。用户可调用 JDBC API 中的 DriverManager、Connection、Statement、ResultSet 类和接口来连接和操纵数据库。其中,DriverManager 是用来管理数据库驱动;Connection 用来建立数据的连接;Statement 是执行 SQL 语言的返回数据到

ResultSet 结果集中,用户就可以在 ResultSet 中使用数据库中的数据;Statement 也可以执行 SQL 语句的修改、添加和删除数据。

本章同时介绍了如何使用 JDBC 技术来实现对数据库的插入,查询,更新,删除操作,并给出了一个典型的实例供给读者学习。

最后本章给出了在数据库开发中文字符的处理,分页的处理,连接池的使用等常用的数据库连接处理方法,这可以帮助读者快速地开展实际的 JSP 数据库开发。

练　习　11

1. 填空题

(1) 一般的 JDBC 对数据库的一次操作流程分为如下 7 个步骤:________、________、________、________、________、________和________。

(2) JDBC 有 4 种数据库的连接方式,它们分别是________、________、________和________。

(3) 分别写出下列 3 种数据库的连接字符串:

MySQL ________________;

SQL Server ________________;

ORACLE ________________。

2. 选择题

(1) 如果在数据库连接中,程序员给错了用户名和密码,JDBC 会抛出什么异常:________

A. ClassNotFoundException　　B. SQLException

C. NullPointerException　　D. AttributeException

(2) 在 JDBC 数据库连接中,若要执行数据库中预存储的存储过程,应该是下列哪个对象?

A. CallableStatement　　B. PreparedStatement

C. Statement　　D. ResultSet

(3) 在 Java 语言中,下列哪个是正确的 JDBC 代码片断。

A.
```
PreparedStatement pstme= con.prepareStatement("insert into EMP
        (EMPNO,ENAME)values(?,?)");
Pstmt.setInt(1,7);
Pstmt.setString(2,"Admin");
```

B.
```
PreparedStatement pstme= con.prepareStatement("insert into EMP
        (EMPNO,ENAME)values(?,?)");
Pstmt.setInt(1,"7");
Pstmt.setString(2,"Admin");
```

C.
```
Statement stmt= con.createStatement("insert into EMP
        (EMPNO,ENAME)values(7,'Admmin'));
```

D.
```
PreparedStatemen stmtl= con.prepareStatement("insertintoEMP
        (EMPNO,ENAME)values(7,'Amin'));
```

3. 实验题

利用数据库连接池的方式改写 11.2.5 中数据库应用实例。

第 12 章　JSP 的 JavaBean 编程

12.1　JavaBean 概述

JavaBean 是用 Java 语言编写的一种可重用的软件组件，也称为 bean。JavaBean 于 1997 年 Sun 的微系统(Sun MicroSystem)提出，它的最主要目标是为第三方的独立软件销售商(Independent Software Vendor)提供 Java 组件，整合组件为软件应用系统。

JavaBean 的功能没有限制，可以根据用户的要求实现特定的功能。一个 Bean 可以完成一个简单的功能，如检查一个文件的拼写，也可以完成复杂功能，如预测一只股票的业绩。JavaBean 可以分成对最终用户是可见的和不可见的组件。对于可见的 JavaBean 组件，用户可以在软件中看到并使用，如使用 Java Swing 包中的大部分 GUI 组件就是 JavaBean 组件，像 javax.swing.JButton 可以创建图形用户界面上的一个按钮。至于不可见的 JavaBean 组件，通常作为业务逻辑处理而出现，如实时多媒体解码软件或用 JavaBean 访问 Web 数据库等。JavaBean 也可以是将现有的一些软件组件或其他的 Java 类转换而来。

从本质上说，JavaBean 也是一种用 Java 语言编写的公共类。Sun 微系统公司规定 JavaBean 是一种特殊性质的公共类，只有一个不带参数的默认构造方法。JavaBean 中预先定义特定的属性和方法模式，通常用 setXXX()和 getXXX()预先定义的行为方法实现对内部属性的设置与访问，也可以自定义的方法实现其他的功能。JavaBean 单独存在没有意义，它是作为具体的应用的一部分。这些特性使得 JavaBean 与一般用 Java 语言定义的 Java 类有着明显区别。对于用户而言，并不需要了解 JavaBean 的内部的实现细节，只需要了解 JavaBean 的具体功能，以及调用哪些方法可以实现这些功能。

当前，JSP 与 JavaBean 的结合已成为 JSP 的主要工作模式之一。JavaBean 的出现，实现了 Java 代码动态内容与 HTML 静态内容的分离，改善了 JSP 页面中不同类型代码相互嵌套的复杂的状况。提高了 JSP 页面的可维护性。从实现功能的角度上来说，JavaBean 作为在 JSP 页面中实现业务逻辑处理的不可见组件在其他多种应用中出现，例如访问 Web 数据库等。这使得实现事物处理的 JavaBean 提高了组件的可重用性。从编程的角度上来说，JavaBean 具有特定的定义模式，编写简单。在应用 JavaBean 方面，可以用 Java 代码直接创建和使用 JavaBean 实例，也可用 JSP 的特定的指令实现对 JavaBean 的创建和使用。这使得应用 JavaBean 非常方便。JavaBean 的这些特点在一定程度上，降低了 JSP 网页的开发人员应用 Java 语言的编程要求。客观上，推广了 JSP 的应用。因此，学习和了解 JavaBean 是必要的。

12.1.1　JavaBean 的简单应用

要应用 JavaBean，必须要定义 JavaBean。JavaBean 编写简单。与编写 Java 类一样，可以使用任何文本编辑软件。只是在保存 JavaBean 时，要将 JavaBean 的类名作为 JavaBean

的文件名，而文件扩展名为“. java”。下列将从一个简单的 JavaBean 例子来了解 JavaBean 的特点。

【例 12-1】 编写一个表示长方体的 JavaBean。具体代码见程序清单 12-1。

程序清单 12-1：

```
//CuboidBean.java
package beans;                                          //定义 beans 包
public class CuboidBean {
    private double width;                               //长方体宽度
    private double height;                              //长方体高度
    private double length;                              //长方体的长度
    private double area;                                //面积
    private double perimeter;                           //周长
    private double volume;                              //体积
    private boolean cuboid;                             //判断是否是合法的长方体
    public double getHeight() {                         //获取高度
        return height;
      }
    public void setHeight(double height) {              //设置高度
        this.height=height;
    }
    public double getLength() {                         //获取长度
        return length;
    }
    public void setLength(double length) {              //设置长度
        this.length=length;
    }
    public double getWidth() {                          //获取宽度
        return width;
    }
    public void setWidth(double width) {                //设置长度
        this.width=width;
    }
    public double getArea() {                           //获取面积
        if(isCuboid())
          area=2*(this.width*this.height+this.width*this.length+this.height*this.length);
        else
          area=0;
        return area;
    }
    public boolean isCuboid() {                         //判断是否是合法的长方体
        if(this.height<=0||this.width<=0||this.length<=0)
            cuboid=false;
        else
            cuboid=true;
```

```
        return cuboid;
    }
    public double getPerimeter() {                           //获取周长
        if(isCuboid())
            perimeter=4* (this.height+this.width+this.length);
        else
          perimeter=0;
        return perimeter;
    }
    public double getVolume() {                              //获取体积
        if(isCuboid())
            volume=this.width* this.height* this.length;
        else
            volume=0;
        return volume;
    }
}
```

在 CuboidBean.java 程序中定义了一个简单的 JavaBean。该 CuboidBean 类是一个公共类，没有定义任何默认构造方法。因此，系统会自动为 CuboidBean 生成一个默认的构造方法。CuboidBean 类用 width、height、length 分别表示宽度属性、高度属性和长度属性，用 area、perimeter 和 volume 分别表示面积、周长和体积属性，用 cuboid 表示是否是合法长方体的属性。然后根据长方体的特性，分别为这些属性定义对应的方法。用 setXXX()的方法作为设置 XXX 对应的属性，而 getXXX()的方法作为获取对应 XXX 对应的属性。对于表示判断的方法，可以用 isXXX()方法实现，如程序代码中 isCuboid()方法。

12.1.2 访问 JavaBean 的基本语法

独立存在的 JavaBean 是没有任何意义的，JavaBean 必须和其他的技术结合才能体现它的作用。JSP 定义了特定的指令和动作来实现 JavaBean 的应用。要应用 JavaBean 实际上是分成了 3 个步骤：

(1) 导入 JavaBean 类，让指定的 JavaBean 类有效；

(2) 用<jsp:useBean>动作创建 JavaBean 实例对象；

(3) 用<jsp:setProperty>或<jsp:getProperty>或直接调用特定对象方法实现对创建对象属性的设置或获取属性的值，达到实现特定的功能的目的。

1. 导入 JavaBean 类

导入 JavaBean 类可以通过 JSP 的页面指令<%@ page>来实现。通过 import 属性指定导入的类。具体的语法如下：

```
<%@page import="类名"%>
```

2. <jsp:useBean>

<jsp:useBean>是创建一个指定范围的 JavaBean 实例对象的动作。只有通过该动作指令，才可以使用 JavaBean。该动作指令的具体语法格式如下：

```
<jsp:useBean id="name" scope="page|request|session|application" 类型模式>
    …//内容实体
</jsp:useBean>
```

在该语法中,有几点说明。

(1) id 属性。id 属性表示创建 JavaBean 对象实例(即 bean)的变量名称,该名称在指定的命名空间中必须是独一无二的,不能出现同名的名称。

(2) scope 属性。scope 属性表示创建 JavaBean 对象实例的作用域,指定 JavaBean 对象实例的生命周期。在 JSP 中只可能为 4 种作用域 page、request、session 和 application 中的一种。具体介绍见 12.2.2 小节。

(3) 类型模式。JavaBean 实例对象的创建必须指定对象所属的类或类别。说明 JavaBean 的类别的类型模型有如下 4 种。在具体应用中,只能使用上述 4 种类型模式中的一种。

① class="类名":指定 JSP 引擎查找 JavaBean 类的路径。

② class="类名" type="类型名":class 属性与 type 属性位置可以互换。对于 type 属性是定义指定 JavaBean 变量的类型。

③ beanName="bean 名字"type="类型名":beanName 属性与 type 属性的位置可以互换。其中,beanName 属性表示一个 bean 的名称,即指定类加载器中 bean 的名称。

④ type="类型名":指定 JavaBean 的类型,按照已定义的类型创建一个对象实例。

3. <jsp:setProperty>

<jsp:setProperty>动作为一个指定 JavaBean 对象(bean)的设置属性的值。具体的语法格式如下:

```
<jsp:setProperty name="bean的名字" 属性表达/>
```

(1) name 属性。name 属性指明 bean 或 JavaBean 实例对象的名字,该名字与用<jsp:useBean>中用 id 属性指定的 bean 名称一致。

(2) 属性表达。<jsp:setProperty>通过设置属性表达来设置具体的属性。<jsp:setProperty>动作中有 4 种形式的属性表达。设置属性时,<jsp:setProperty>中 property 指定的属性必须和 bean 中的 setXXX()方法中的属性 XXX 一致。

① property="*":表示为已经创建的 bean 的所有属性设置或修改属性值。

② property="属性名":表示为指定属性设置或修改属性值。

③ property="属性名"param="参数名":表示为指定属性设置或修改属性值。param 属性表示用户请求的参数,此时,属性值来自于用户请求参数对应的值,通常请求参数来自于 Web 表单。这种使用方法通常用于属性的名称与 request 的参数名称不一致的情况。

④ property="属性名" value="字符串|<%=表达式%>":表示设置指定属性的属性值。该属性值可以是字符串,在使用 JavaBean 对象时,字符串会根据 JavaBean 属性的类型转换成对应的类型。也可以是以"<%=表达式%>"形式出现,表达式可以是数学表达式,也可以是方法的调用等多种形式。

【例 12-2】 用<jsp:setProperty>实现 CuboidBean 的访问,计算一个长方体的周长、面积和体积。为了实现具体代码见程序清单 12-2,执行结果如图 12-1 和图 12-2 所示。

程序清单 12-2：

```
<!--JSP12-2.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"
import="beans.CuboidBean"% >
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>长方体信息输入</title>
</head>
<body>
  <p>
    <form action=" " method="post"><!--定义表单-->
     <table border="1">
         <caption>输入长方体的长、宽、高</caption>
         <tr>
            <td>长度</td>
            <td><input type="text" name="length"/></td>
         </tr>
         <tr>
            <td>宽度</td>
            <td><input type="text" name="width"/></td>
         </tr>
         <tr>
            <td>高度</td>
            <td><input type="text" name="height"/></td>
         </tr>
         <tr>
            <td><input type="submit" value="提交"/></td>
         </tr>
     </table>
    </form>
  </p>
  <p>
<jsp:useBean id="cuboid1" scope="page" class="beans.CuboidBean"/>
<!--创建实例对象 cuboid1-->
<jsp:setProperty name="cuboid1" property="*"/><!--用表单的数据设置属性-->
   <%
      if(cuboid1.isCuboid()){
      out.println("CuboidBean的应用—<br> ");
      out.println("长="+cuboid1.getLength());
      out.println("宽="+cuboid1.getWidth());
```

```
        out.println("高="+cuboid1.getHeight());
        out.println("周长："+cuboid1.getPerimeter());
        out.println("面积："+cuboid1.getArea());
        out.println("体积："+cuboid1.getVolume());
       }
      %>
<jsp:useBean id="cuboid2" scope="page" class="beans.CuboidBean">
<jsp:setProperty name="cuboid2" property="width" value="23"/><!--修改属性值-->
<jsp:setProperty name="cuboid2" property="height"/>
<jsp:setProperty name="cuboid2" property="length"/>
</jsp:useBean>
</p>
<p>
  <%
    if(cuboid2.isCuboid()){
    out.println("CuboidBean的应用二<br>");
    out.println("长="+cuboid2.getLength());
    out.println("宽="+cuboid2.getWidth());
    out.println("高="+cuboid2.getHeight());
    }
  %>
</p>
<p>
  <jsp:useBean id="cuboid3" scope="page" class="beans.CuboidBean">
   <jsp:setProperty name="cuboid3" property="width"
     value="<%=cuboid2.getHeight()%>"/><!--用表达式修改属性值-->
   <jsp:setProperty name="cuboid3" property="height"
     value="<%=cuboid2.getLength()%>"/>
   <jsp:setProperty name="cuboid3" property="length"
     value="<%=cuboid2.getWidth()%>"/>
  </jsp:useBean>
  <%
    if(cuboid3.isCuboid()){
    out.println("CuboidBean的应用三->修改 CuboidBean的长宽高<br>");
    out.println("长="+cuboid3.getLength());
    out.println("宽="+cuboid3.getWidth());
    out.println("高="+cuboid3.getHeight());
   }
  %>
</p>
</body>
</html>
```

对例 12-2 有几点说明如下。



图 12-1　输入表单数据

长方体信息输入 - Microsoft Internet Explorer

地址(D) http://localhost:8080/chenyi/chapter12/JSP12-1.jsp

输入长方体的长、宽、高

长度

宽度

高度

提交

CuboidBean的应用一
长=3.0 宽=4.0 高=5.0 周长：48.0 面积:94.0 体积：60.0

CuboidBean的应用二
长=3.0 宽=23.0 高=5.0

CuboidBean的应用三->修改CuboidBean的长宽高
长=23.0 宽=5.0 高=3.0

图 12-2　应用 CuboidBean 的对象

首先用<%@ page>指令的 import 属性指定导入的 JavaBean 的类“beans.CuboidBean”。请注意，要实现导入，“beans.CuboidBean”与其他 JavaBean 的类一样，必须放置在 Tomcat 服务器对应 Web 应用的“WEB-INF/classes”目录下。否则 JSP 引擎将无法查找到该 JavaBean 类。

其次，通常用表单实现数据的输入。为此，表单中各个输入组件名必须与 JavaBean 的属性名一致，即表单中的输入组件名，如 length 必须与 CuboidBean 的实例对象的属性名 length 一致。但是，如果表单的输入组件名与 JavaBean 实例对象的属性名不一致，则需要<jsp:setProperty>中用 param 属性指定表单的控件名，即请求的参数名，对应于特定的属

性。假设表单中长度的输入文本框组件定义为："<input type="text" name="len"/>"，则在设置 CuboidBean 的实例对象的 length 的属性时必须写成："<jsp:setProperty name="cuboid1" property="length" param="len"/>"。

此外，在程序清单 12-2 中定义了 3 个不同的 beans.CuboidBean 的实例对象：cuboid1、cuboid2 和 cuboid3。它们的作用虽然同为 page，是从定义处到文件末有效。但是，这 3 个实例对象的数据可以不同。

4. <jsp:getProperty>

在上个例子中，是通过形如"JavaBean 对象实例.getXXX()"方法来获取 JavaBean 对象实例的属性值。这种做法无疑要求对 bean 的方法有深入的了解。JSP 中还提供了<jsp:getProperty>动作来实现获取 bean 的属性值。<jsp:getProperty>的语法格式如下：

```
<jsp:getPropety name="对象名" property="属性名"/>
```

其中，name 属性指定 JavaBean 的实例对象名，而 property 属性指定 JavaBean 实例对象的属性。通过这样的一个动作，实现获取指定 JavaBean 对象的属性值，使用形式简单。

【例 12-3】 用<jsp:getProperty>实现 CuboidBean 的访问，计算一个长方体的周长、面积和体积，执行结果如图 12-3 所示。

程序清单 12-3：

```
<!--JSP12-3.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java" import="beans.CuboidBean"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>长方体信息输入</title>
</head>
<body>
  <p>
    <form action="JSP12-3.jsp" method="post">
     <table border="1">
         <caption>输入长方体的长、宽、高</caption>
         <tr>
             <td>长度</td>
             <td><input type="text" name="length"/></td>
         </tr>
         <tr>
             <td>宽度</td>
             <td><input type="text" name="width"/></td>
         </tr>
         <tr>
             <td>高度</td>
             <td><input type="text" name="height"/></td>
         </tr>
```

```
            <tr>
                <td><input type="submit" value="提交"/></td>
            </tr>
        </table>
      </form>
    </p>
    <p>
<jsp:useBean id="cuboid" scope="page" class="beans.CuboidBean"/>
<jsp:setProperty name="cuboid" property="*"/>
长=<jsp:getProperty name="cuboid" property="length"/><br/>
高=<jsp:getProperty name="cuboid" property="height"/><br/>
宽=<jsp:getProperty name="cuboid" property="width"/><br/>
周长=<jsp:getProperty name="cuboid" property="perimeter"/><br/>
面积=<jsp:getProperty name="cuboid" property="area"/><br/>
体积=<jsp:getProperty name="cuboid" property="volume"/>
</p>
</body>
</html>
```



图 12-3　jsp:getProperty 的应用实例

12.2　JSP 页面使用 JavaBean

在上一节中，了解了 JavaBean 的基本概念以及简单应用 JavaBean。在本节中，将深入了解 JSP 使用 JavaBean，具体涉及 JavaBean 的属性、JavaBean 的作用范围。

12.2.1 JavaBean 的属性

JavaBean 的属性体现了 JavaBean 内部特征，是 JavaBean 内部状态的抽象表示。JavaBean 属性并不只是在内部表现 JavaBean 的特征，更重要的是，用户根据具体应用的不同，通过设置或获取属性，实现特定的功能。JavaBean 的属性可以表现出不同的形式，对 JavaBean 的实际应用产生不同的效果。根据 JavaBean 规范，按照属性形式的不同，可以将属性分成 4 种类型：简单属性(Simple)、索引属性(Indexed)、绑定属性(Bound)和约束属性(Constrained)。

1. 简单属性(Simple)

简单属性是通过 JavaBean 对象的相关方法的调用来实现对 JavaBean 属性的访问。一般是针对单值属性而言，JavaBean 的属性只对应一个值。在这种情况下，JavaBean 的属性定义 setXXX()方法实现设置属性，而用 getXXX()获取属性。具体形式如下：

```
void set<Property> (PropertyType value)          //设置属性
PropertyType get<Property> ()                    //获取属性
```

如果属性所属的数据类型是布尔类型，一般表示判断。这时，可以考虑使用 isXXX()方法来获取属性的值，仍用 setXXX()来设置属性。具体形式如下：

```
void set<Property> (boolean value)          //设置属性
boolean is<Property> ()                     //获取判断的结果
```

使用属性，可通过调用 JavaBean 的实例对象的对应方法来实现。如例 12-2 的程序清单 12-2 中就展示了一个简单属性 JavaBean 属性的实例。其中调用对象的方法可以通过直接用对象名实现，如 cuboid1. getLength()，也可以通过动作<jsp: setProperty>和<jsp:getPropety>来实现。访问形式简单、方便。

2. 索引属性(Indexed)

不同于简单属性，索引属性的 JavaBean 属性对应的是指定范围一组值，通常用数组定义。要访问属性中一组值中的一个必须通过索引实现。索引必须是 Java 语言中的整型(int)。索引属性的 JavaBean 属性的访问有多种形式：可以设置属性的一组值，也可以设置属性索引指定的单值；可以是获取属性的一组值，也可以是获取指定属性索引指向的单值。具体形式如下：

```
void set<Property> (int index,<PropertyType> value);          //索引设置
void set<Property> (<PropertyType[]> value);                  //数组设置
void PropertyType get<Property> (int index);                  //索引获取
void PropertyType[] get<Property> ();                         //数组获取
```

【例 12-4】 一个简单索引属性 JavaBean 属性的应用实例。GradeBean 定义见程序清单 12-4，应用 GradeBean 的 JSP 页面见程序清单 12-5，执行结果如图 12-4 所示。

程序清单 12-4：

```
//GradeBean.java
package beans;
```

```java
public class GradeBean {
    private String[] grade={"大学一年级","大学二年级","大学三年级","大学四年级"};
    public String[] getGrade() {                              //获取数组
     return grade;
    }
    public void setGrade(String[] grade) {                    //设置数组
     this.grade=grade;
    }
    public String getIndexGrade(int index) {                  //获取索引
       if(isValidIndex(index))
           return this.grade[index];
       return null;
    }
    public void setIndexGrade(int index,String str){          //设置索引
        if(isValidIndex(index))
            this.grade[index]=str;
    }
    public boolean isValidIndex(int index){
        if(index<this.grade.length&&index>=0)
            return true;
        return false;
    }
}
```

程序清单 12-5：

```jsp
<!--JSP12-5.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java" import="beans.GradeBean"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>索引属性属性的应用</title>
</head>
<body>
  <p>
  <jsp:useBean id="grade" scope="page" class="beans.GradeBean">
   <jsp:setProperty name="grade" property="*"/>
  </jsp:useBean>
  <%
     String gradeNames1[]=grade.getGrade();
     out.println("应用一：<br>");
     for(int i=0;i<gradeNames1.length;i++)
      out.println(grade.getIndexGrade(i)+"<br>");     //依次访问索引指向的元素
     out.println("<br> 应用二：<br>");
     String gradeNames2[]={"研究生一年级","研究生二年级","研究生三年级"};
     grade.setGrade(gradeNames2);                     //重新设置数组
```

```
        for(int j=0;j<gradeNames2.length;j++)
          out.println(grade.getIndexGrade(j)+"<br> ");
    %>
  </p>
  </body>
  </html>
```



图 12-4　索引型属性的应用

3. 绑定属性(Bound)和约束属性(Contrainted)

绑定属性是在 JavaBean 实例对象的属性值发生变化时，就要发送一个 PropertyChange 事件通知所有的相关的监视器，然后进行相关的处理。要实现这样的一个 JavaBean 属性，需要定义 addPropertyChangeListener()和 removePropertyChangeListener()，用它们分别表示增加属性变化监听器和移除属性变化监听器。

约束属性是 JavaBean 的属性受到约束。这种约束是由监听器来决定是否要否决属性发生的任何一个变化，强迫属性返回原有的设置。

具有绑定属性和约束属性的 JavaBean 大量出现在图形组件的定义和使用上，在本节将对它们不做介绍。

12.2.2　JavaBean 的作用域

通过<jsp:useBean>动作的 scope 属性，实现设置 JavaBean 的作用域。JavaBean 的作用域设定了 JavaBean 的应用范围，这使得 JavaBean 对象在服务器具有生存周期。JSP 中规定 JavaBean 的只有 4 种作用域：page、request、session 和 application。为了说明 JavaBean 的作用域，本节定义了一个计数器的 JavaBean，命名为 CounterBean，通过它来了解 JavaBean 在不同作用域中不同的特点。具体代码见程序清单 12-6。

程序清单 12-6：

```
//CounterBean.java
package beans;
```

```
public class CounterBean {
     private int counter;
public int getCounter() {                          //获取访问次数
        return counter;
}
public void setCounter(int counter) {              //设置访问次数
        this.counter=counter;
}
}
```

1. page 作用域

page 作用域表示创建的对象只在当前 JSP 页面中有用。如果关闭当前 JSP 页面，使得 JSP 页面失效，这时创建的 JavaBean 对象将会注销。但是，如果重新打开一个 JSP 页面，服务器会重新创建一个新的 JavaBean 对象。下列程序清单 12-7 是使用了一个作用域为 page 的 JavaBean 对象，执行结果如图 12-5 所示。可以发现无论是刷新程序清单定义的 JSP12-7.jsp 页面，还是在浏览器中重新打开一个新的 JSP12-7.jsp 文件，显示的结果不会发生变化，JavaBean 对象 counter 的属性 counter 为 1。

程序清单 12-7：

```
<!--JSP12-7.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java" import="beans.CounterBean"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>索引属性属性的应用</title>
</head>
<body>
  <p>
  <jsp:useBean id="counter" scope="page"
     class="beans.CounterBean"/>
  <%
     out.println("访问次数为：");
  %>
    <jsp:setProperty name="counter" property="counter"
     value="<%=counter.getCounter()+1%>"/><!--修改访问次数-->
    <jsp:getProperty name="counter" property="counter"/><!--获取访问次数;-->
</p>
</body>
</html>
```

2. request 作用域

request 作用域表示创建的对象在用户的本次请求中有效。具体作用于当前的 JSP 页面，以及与当前 JSP 页面共享同一个请求的页面，包含了当前 JSP 页面中用<%@ include>指令以及运用<jsp:forward>动作导向的其他 JSP 页面。当服务器响应服务端请求后，JavaBean 的对象会失去作用。



图 12-5 page 作用域的应用

下列的程序中定义了一个作用域为 request 的 JavaBean 对象。由于 JSP12-8.jsp 页面中包含了一个<jsp:forward>动作导向 JSP12-9.jsp 页面。这使得 counter 对象的作用于整个用户请求。从运行结果可以发现，这时的访问次数为 2。但是，用户重新请求，counter 对象会重新创建。所以，无论刷新页面，还是重新打开一个新的浏览页面，访问次数不会变化，执行结果如图 12-6 所示。

程序清单 12-8：

```
<!--JSP12-8.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"
import="beans.CounterBean"%>
<html>
<head>
<title> 访问次数统计应用二：</title>
</head>
<body>
  <p>
  <jsp:useBean id="counter" scope="request" class="beans.CounterBean"/>
  <%
     out.println("访问次数为：");
  %>
   <jsp:setProperty name="counter" property="counter"
    value="<%=counter.getCounter()+1%>"/>                <!--修改访问次数-->
   <jsp:getProperty name="counter" property="counter"/>   <!--获取访问次数;-->
   <jsp:forward page="JSP12-9.jsp"/>
</p>
</body>
</html>
```

程序清单 12-9：

```
<!--JSP12-9.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java" import="beans.CounterBean"%>
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>JSP12-19的页面：</title>
</head>
<body>
  <p>
  <jsp:useBean id="counter" scope="request" class="beans.CounterBean"/>
  <%
     out.println("现在访问次数为：");
  %>
   <jsp:setProperty name="counter" property="counter"
     value="<%=counter.getCounter()+1%>"/>
   <jsp:getProperty name="counter" property="counter"/>
</p>
</body>
</html>
```



图 12-6　request 作用域的应用

3. session 作用域

session 会话是一段时间内，用户与 Web 服务器交互的过程。而 session 作用域表示在整个 session 会话过程中有效，除非 session 注销。比如 session 会话过期，或关闭 Web 浏览器，或从会话过程中主动退出。当 Web 服务器创建了 JavaBean 对象后，这个对象将在整个 session 会话过程有效，并且 Web 服务器不会为该用户创建其他的 JavaBean 对象。可以通过内置对象 session 调用 getAttribute()方法，形如“(JavaBean 类型)session. getAttribute(JavaBean 对象名)来获取指定的 JavaBean 对象。

程序清单 12-10 是使用了一个作用域为 session 的 JavaBean 对象，执行结果如图 12-7 所示。可以发现，只要不断地刷新 JSP 页面，显示的访问次数会累次增加。但是，打开浏览器访问 JSP12-10. jsp 文件，Web 服务器会创建一个新的 counter 对象，访问次数又重新从 1 开始。

程序清单 12-10：

```
<!--JSP12-10.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java" import="beans.CounterBean"%>
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>访问次数统计应用三：</title>
</head>

<body>
  <p>
  <jsp:useBean id="counter" scope="session" class="beans.CounterBean"/>
  <%
     out.println("访问次数为：");
  %>
   <jsp:setProperty name="counter" property="counter"
    value="<%=counter.getCounter()+1%>"/>
   <!--修改访问次数-->
   <jsp:getProperty name="counter" property="counter"/>
   <!--获取访问次数;-->
</p>
</body>
</html>
```



图 12-7 session 作用域的应用

4. application 作用域

application 作用域是应用范围最广，表示在 Web 应用的整个过程有效。Web 应用中的所有 JSP 页面都可以使用同一个 JavaBean 对象。这个针对所有 Web 应用的 JavaBean 对象可以通过内置对象 application 的 getAttribute()方法来获取，具体形式如“(JavaBean 类型)application.getAttribute(JavaBean 对象名)”。

在程序清单 12-11 中，定义了作用域为 application 的 JavaBean 对象 counter，并通过 application 来访问该对象的一个复本，运行结果如图 12-8 所示。从运行结果可以发现，不断刷新页面或打开浏览器重新访问 JSP12-11.jsp 文件，访问次数不断增加，这是因为在整个 Web 应用中，Web 服务器只有一个有效的 counter 对象。

程序清单 12-11：

```
<!--JSP12-11.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java" import="beans.CounterBean"%>
<html>
<head>
<title>访问次数统计应用四:</title>
</head>
<body>
  <p>
  <jsp:useBean id="counter" scope="application" class="beans.CounterBean"/>
   <jsp:setProperty name="counter" property="counter"
      value="<% =counter.getCounter()+1%>"/>    <!--修改访问次数-->
   <%
    CounterBean count=(CounterBean)application.getAttribute("counter");
    //利用 application 对象,获取 counter 对象的复本
    out.println("访问次数为:"+ count.getCounter());    //输出访问次数
   %>
   </p>
</body>
</html>
```



图 12-8 application 作用域的应用

12.3 利用 JavaBean 访问数据库

数据库是实现动态网页的重要内容。第 11 章介绍了 JSP 访问 Web 数据库的关键技术。本节将不再对这些技术做深入了解。本节侧重了解如何用 JavaBean 来实现数据库的连接和数据库的访问。通过 JavaBean 实现对数据库的访问,有效地将数据库处理逻辑与静态页面分开,有利于用 JSP 实现动态内容。

JSP 中连接数据库是 JDBC 技术。JDBC 技术被当前许多数据库系统支持,大部分具有 JDBC 的驱动程序。利用 JDBC 技术访问 Web 数据库一般有 7 个步骤:

(1) 导入 JDBC 类;

(2) 装载/注册驱动程序;

（3）连接数据库；
（4）创建语句对象；
（5）执行 SQL 语句；
（6）处理结果；
（7）关闭连接。

根据这 7 个步骤，可以定义可实现连接和关闭功能的 JavaBean 和执行访问数据库数据的 JavaBean。

为了说明 JavaBean 访问数据库，在本节中用 JSP 和 JavaBean 定义一个简单的通讯录应用实例。要求该通讯录可以实现查询、添加、删除、修改。首先，用 MySql 定义一个通讯录数据表 addresslist，addresslist 的定义的数据项有 addr_Id（通信编号）、name（用户名）、phone（联系电话）、address（通信地址）、email（电子邮件）。

程序清单 12-12：

```
/* addressDB.sql */
create database addressDB;
use addressDB;
create table addresslist(
      addr_Id int PRIMARY KEY,
      name varchar(20),
      phone varchar(20),
      address varchar(100),
      email varchar(40)
      );
```

其次，为了操作方便，定义一个 JavaBean 表示通信录的每一个记录。具体代码见程序清单 12-13。

程序清单 12-13：

```
//AddressBean.java
package beans;
public class AddressBean {
   private int addr_Id;                         //表示编号；
   private String name;                         //用户名
   private String phone;                        //联系电话
   private String address;                      //通信地址；
   private String email;                        //电子邮件；
   public int getAddr_Id() {
      return addr_Id;
   }
   public void setAddr_Id(int addr_Id) {
      this.addr_Id=addr_Id;
   }
   public String getAddress() {
      return address;
```

```
    }
    public void setAddress(String address) {
        this.address=address;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email=email;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name=name;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone=phone;
    }
}
```

12.3.1 JavaBean 连接数据库

要定义 JavaBean 连接数据库。首先，必须要确定数据库系统的驱动程序。通过驱动程序才可以装载和注册数据库；其次，在连接数据库时，需要提供访问数据库的相关信息，如数据库的 URL、用户名和密码等；最后，为了数据库的正常关闭，需要执行关闭数据库的功能。根据这些基本要求，可以定义一个通用的连接数据库的 JavaBean。

下列程序清单 12-14 是一个通用的连接数据库的 JavaBean，命名为 ConnBean.java。

程序清单 12-14：

```
//ConnBean.java
package beans;
import java.sql.*;                                              //导入 jdbc
public class ConnBean {
    private String driver="com.mysql.jdbc.Driver";              //默认数据库为 MySql
    private String jdbcurl="jdbc:mysql://localhost:3306/";      //jdbcurl
    private String database="addressDB";                        //数据库或数据源
    private String userName="";                                 //用户名
    private String password="";                     //密码
    private Connection connection=null;             //建立数据库的连接
    public ConnBean(){                              //自定义默认构造方法
      getConnection();
```

```
}
public Connection getConnection() {                 //获取建立连接的 connection
try{
      Class.forName(driver);                        //注册驱动程序
      connection= DriverManager.getConnection(jdbcurl+ database,username,password);
                                                    //建立连接
  }catch(ClassNotFoundException e1){
   e1.printStackTrace();
  }catch(SQLException e2){
   e2.printStackTrace();
  }
  return connection;
}
public void closeConnection(){                      //关闭连接
try{
      if(connection!=null)
           connection.close();
         connection=null;
}catch(SQLException e3){
   e3.printStackTrace();
}
}
public String getDriver() {                         //获取驱动程序名
  return driver;
}
public void setDriver(String driver) {              //设置驱动程序名
  this.driver=driver;
}
public String getDatabase() {                       //获取数据库/数据源名
  return database;
}
public void setDatabase(String database) {          //设置数据库/数据源名
  this.database=database;
}
public String getPassword() {                       //获取访问数据库的密码
  return password;
}
public void setPassword(String password) {          //设置访问数据库的密码
  this.password=password;
}
public String getJdbcurl() {                        //获取 jdbcURL 的协议部分
  return jdbcurl;
}
public void setJdbcurl(String url) {                //设置 jdbcURL 的协议部分
  this.jdbcurl=url;
```

```
    }
    public String getUserName() {                    //获取访问数据库的用户名
        return userName;
    }
    public void setUserName(String userName) {       //设置访问数据库的用户名
        this.userName= userName;
    }
}
```

上例的 ConnBean.java 是具有简单属性属性的 JavaBean 代码。它实现了加载驱动程序、连接数据库、关闭数据库,以及设置和获取关于连接数据库的相关数据,如 jdbcURL、、访问数据库的用户名和访问数据库的密码等。为了简化处理,基于不同数据库系统连接的不同,程序清单 12-12 中,JDBC URL 并不是一次定义,而是通过属性 jdbcurl 和 database 组合来实现。属性 jdbcurl 代表的是访问数据库的协议与子协议,而 database 表示数据库,或数据源(采用 JDBC-ODBC 桥接方式)。上述的 ConnBean.java 在连接数据库具有一定的通用性。

12.3.2 JavaBean 实现数据库操作

连接数据库的目的是实现对数据库的操作和访问。在第 11 章中,JSP 页面操作数据库处理过程复杂。要实现数据库的具体操作,需要获取 java. sql. Statement、java. sql. PerpareStatement 或 java. sql. CallableStatement 的执行语句对象,通过这些执行语句对象调用相应的执行方法,如 execute()、excuteQuery()等执行对数据库的操作。也可以获取 java. sql. ResultSet 的结果集对象,让用户获得执行操作后的结果。

接下来,将通过对通讯录的应用实例着手,来了解可以执行一个数据库操作的 JavaBean。要执行数据库操作离不开对数据库的连接。所以,定义一个 AddressDBean 的执行数据库的 Bean,它是 ConnBean 的子类。具体内容将程序清单 12-15。

程序清单 12-15:

```
//AddressDBean.java
package beans;
import java.sql.* ;
import java.util.* ;
public class AddressDBean extends ConnBean{
    private Connection connection= null;
    public AddressBean[] getAllRecords(){                //获取所有记录
            ResultSet rs= null;
            PreparedStatement pstmt= null;
            Collection list= new ArrayList();
            try{ connection= getConnection();
                pstmt= connection.prepareStatement("select *  from addresslist");
                                                         //数据表 addresslist
                rs= pstmt.executeQuery();
                while(rs.next()){
```

```
                AddressBean address=new AddressBean();
                address.setAddr_Id(rs.getInt(1));
                address.setName(rs.getString(2));
                address.setPhone(rs.getString(3));
                address.setAddress(rs.getString(4));
                address.setEmail(rs.getString(5));
                list.add(address);
            }
         }catch(SQLException e){
                e.printStackTrace();
            }finally{
                closePstmt(pstmt);
                closeConnection(connection);
            }
          AddressBean[] records=(AddressBean[])list.toArray(new AddressBean[0]);
            return records;
    }
    public boolean insertRecord(AddressBean record){        //插入记录
        PreparedStatement pstmt=null;
        String insStr="insert into addresslist values (?,?,?,?,?)";
        if(record==null) return false;
        try{
            connection=getConnection();
            pstmt=connection.prepareStatement(insStr);
            pstmt.setInt(1, record.getAddr_Id());
            pstmt.setString(2, record.getName());
            pstmt.setString(3, record.getPhone());
            pstmt.setString(4, record.getAddress());
            pstmt.setString(5, record.getEmail());
            pstmt.execute();
         }catch(SQLException e){
             e.printStackTrace();
         }finally{
             closePstmt(pstmt);
             closeConnection(connection);;
         }
         return true;
    }
    public boolean deleteRecord(AddressBean record){        //删除记录;
        PreparedStatement pstmt=null;
        String delStr="delete from addresslist where address_id=?";
        if(record==null) return false;
        try{
            getConnection();
            pstmt=connection.prepareStatement(delStr);
```

```
                pstmt.setInt(1, record.getAddr_Id());
                pstmt.execute();
            }catch(SQLException e){
                e.printStackTrace();
            }finally{
                closePstmt(pstmt);
                closeConnection(connection);
            }
            return true;
        }
        public boolean updateRecord(AddressBean ro,AddressBean rn){                    //修改记录
            if(ro==null||rn==null) return false;
            PreparedStatement pstmt=null;
            String updStr="update addresslist set name=?,phone=?,address=?,email=?";
            updStr=updStr+" where addr_id="+ro.getAddr_Id();
            try{
                connection=getConnection();
                pstmt=connection.prepareStatement(updStr);
                pstmt.setString(1,rn.getName());
                pstmt.setString(2, rn.getPhone());
                pstmt.setString(3, rn.getAddress());
                pstmt.setString(4, rn.getEmail());
                pstmt.executeUpdate();
            }catch(SQLException e){
                e.printStackTrace();
            }finally{
                closePstmt(pstmt);
                closeConnection(connection);
            }
            return true;
        }
}
```

在通讯录应用中，AddressDBean 仍是一个简单属性的 JavaBean，由于继承了 ConnBean，它不但具有连接功能，还具有数据库访问的功能。在实际应用中，也可以根据实际需要定义新的执行数据库访问的方法。

12.3.3 访问数据的应用实例

JavaBean 可以实现连接和访问数据库。访问数据库的具体操作涉及查询数据、插入数据、删除数据和修改数据。在本节中，利用 AddressDBean 来实现数据库访问的前 3 种基本功能。

1. 查询通讯录的所有数据

查询通讯录的所有数据，具体内容见程序清单 12-16，执行结果如图 12-9 所示。

程序清单 12-16：

```
<!--JSP12-16.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="beans.*,java.sql.*"%>
<html>
<head>
<title>通讯录应用实例->显示所有的记录</title>
</head>
<body>
  <jsp:useBean id="conn" scope="session" class="beans.AddressDBean"/>
  <table border="1">
     <caption>通讯录</caption>
     <tr>
        <th>编号</th>
        <th>姓名</th>
        <th>联系电话</th>
        <th>通信地址</th>
        <th>E-mail</th>
     </tr>
     <%
      AddressBean[] records=conn.getAllRecords();             //获取所有记录
      if(records!=null){
          for(int i=0;i<records.length;i++){                  //显示记录
            out.println("<tr> ");
            out.println("<td> "+records[i].getAddr_Id()+"</td> ");
            out.println("<td> "+records[i].getName()+"</td> ");
            out.println("<td> "+records[i].getPhone()+"</td> ");
            out.println("<td> "+records[i].getAddress()+"</td> ");
            out.println("<td> "+records[i].getEmail()+"</td> ");
            out.println("</tr> ");
          }
        }
     %>
  </table>
</body>
</html>
```

2. 插入数据

为通讯录增加新的数据，具体内容见程序清单 12-17，执行结果如图 12-10 所示。

程序清单 12-17：

```
<!--JSP12-17.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="beans.AddressBean,beans.ConnBean"%>
<html>
<head>
<title>通讯录应用二->添加记录</title>
```



图 12-9　显示所有数据

```
</head>
<body>
  <p>
   <table>
     <form action="" method="post">
       <caption>添加新记录</caption>
       <tr>
         <td>编号</td>
         <td><input type="text" name="addr_Id"/> * </td>
       </tr>
       <tr>
         <td>姓名</td>
         <td><input type="text" name="name"/> * </td>
       </tr>
       <tr>
         <td>电话号码</td>
         <td><input type="text" name="phone"/></td>
       </tr>
       <tr>
         <td>通信地址</td>
         <td><textarea name="address" cols="20" rows="5"></textarea></td>
       </tr>
       <tr>
         <td>E-mail</td>
         <td><input type="text" name="email"/> * </td>
       </tr>
       <tr>
         <td/>
         <td><input type="submit" value="增加"/>
```

```
            <input type="reset" value="重写"/>
          </td>
        </tr>
      </form>
    </table>
    <jsp:useBean id="addr" scope="session" class="beans.AddressBean"/>
      <jsp:setProperty name="addr" property="*"/>
    <jsp:useBean id="conn" scope="session" class="beans.AddressDBean"/>
    <%
      if(addr.getAddr_Id()!=0&&conn.insertRecord(addr))          //插入记录
        out.println("<hr>添加记录成功");
    %>
  </p>
</body>
</html>
```



图 12-10　插入数据

3. 删除数据

删除通讯录的一条记录，具体内容见程序清单 12-18，执行结果如图 12-11 所示。

程序清单 12-18：

```
<!--JSP12-18.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="beans.AddressBean,beans.ConnBean"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>通讯录应用三->删除记录</title>
</head>
```

```
<body>
  <p>
  <jsp:useBean id="conn" scope="session" class="beans.AddressDBean"/>
  <form action="" method="post">
  <table border="1">
      <caption>通讯录</caption>
      <tr><th> 编号</th>
          <th>姓名</th>
          <th>联系电话</th>
          <th>通信地址</th>
          <th>E-mail</th>
          <th>选择</th>
      </tr>
      <%
       AddressBean[] records=conn.getAllRecords();
       if(records!=null){
           for(int i=0;i<records.length;i++){
             out.println("<tr>");
             out.println("<td>"+records[i].getAddr_Id()+"</td>");
             out.println("<td>"+records[i].getName()+"</td>");
             out.println("<td>"+records[i].getPhone()+"</td>");
             out.println("<td>"+records[i].getAddress()+"</td>");
             out.println("<td>"+records[i].getEmail()+"</td>");
      %>
           <td><input type="checkbox" name="<%="check"+i%>" value="<%=i%>" />
           </td>
      <% out.println("</tr> ");
           }
         }
      %>
    </table>
    <input type="submit" value="提交"/>
    </form>
    <%
      String[] check=new String[records.length];
      for(int i=0;i<check.length;i++){
        check[i]=request.getParameter("check"+i);
        if(check[i]==null)
          check[i]="";
          if(check[i].equals(""+i)){
            if(conn.deleteRecord(records[i])){                    //删除记录
              out.println("<hr>删除成功");
              response.setHeader("refresh","1");
            }
          }
      }
    %>
```

```
  </p>
</body>
</html>
```



图 12-11　删除数据界面

小　　结

JavaBean 是一个可重用的组件，它具有简单、利于维护、使用方便。为此，JavaBean 得到广泛应用。与 JSP 结合，JavaBean 有效实现了页面的静态内容与动态内容的分离。本章详细介绍了 JavaBean 的基本概念，JSP 应用 JavaBean 的基本语法、应用于 JSP 中常见的 JavaBean 的属性形式、JavaBean 在 JSP 页面中的作用域，以及 JSP 如何利用 JavaBean 实现数据库的连接和访问。并通过具体的应用实例，对这些内容做了详细的说明。

练　习　12

1. 填空题

(1) JavaBean 的属性有________、________、________和________。

(2) JavaBean 是________。

(3) JavaBean 在 JSP 的作用域有________、________、________和________。

(4) JSP 动作________可以实现 JavaBean 属性的设置。

(5) JSP 动作________可以实现 JavaBean 属性的获取。

2. 选择题

(1) 下列说法错误的是________。

A. JSP 中只能用<jsp:setProperty>动作来调用 JavaBean 的属性值

B. JSP 可以用<jsp:setProperty>动作设置索引属性的 JavaBean 属性

C. JSP 必须先导入 JavaBean 类，才可以创建 JavaBean 的对象

D. JSP 中<jsp:useBean>动作用来创建 JavaBean 的实例对象

E. <jsp:useBean>动作是通过属性 scope 来设置 JavaBean 的作用

(2) 利用程序清单 12-6 的 CounterBean，与下列的部分程序段，该部分程序中对 counter 描述正确的是：

```
⋮
<jsp:useBean id="counter" scope="request" class="beans.CounterBean"/>
<%     out.println("访问次数为：");    %>
<jsp:setProperty name="counter" property="counter"
value="<%=counter.getCounter()+1%>"/>
<jsp:getProperty name="counter" property="counter"/>
<jsp:forward page="other.jsp"/>
⋮
```

A. counter 对象在可以为所有 Web 应用共享。

B. 关闭浏览页面后，counter 对象立即消失。

C. counter 对象在用户的整个会话交互过程有效。

D. counter 对象在 Web 服务器响应客户请求后消失。

E. counter 对象在会话过期后才消失。

(3) 利用程序清单 12-6 的 CounterBean，下列的 JSP 程序存在错误的是第________行。

```
第 1行    <%@page import="beans.CounterBean"%>
第 2行    <html>
第 3行    <body>
第 4行    <p>
第 5行    <jsp:useBean id="counter" scope="request" class=" CounterBean"/>
第 6行    <%   out.println("访问次数为：");%>
第 7行    <jsp:setProperty name="counter" property="counter"
第 8行    value="<%=counter.getCounter()+1%>"/>
第 9行    <!--修改访问次数-->
第 10行   <jsp:getProperty name="counter" property="counter"/>
第 11行   <!--获取访问次数;-->
第 12行   <jsp:forward page="JSP12-9.jsp"/>
第 13行   </p>
第 14行   </body>
第 15行   </html>
```

A. 第 1 行　　B. 第 5 行　　C. 第 7 行　　D. 第 8 行

3. 实验题

设计一个通用的 JavaBean，实现通讯录的所有记录分页显示。

第 13 章　JSP 的 Servlet 编程

13.1　Servlet 技术

Servlet 是用 Java 语言编写服务器的程序，用于服务器端的互联网应用。是 Sun 公司回应 CGI 编程技术而产生的新技术。早在 1995 年，Sun 公司的 James Gosling 就开始构思 Servlet 理念。但他没有继续研究这个新概念。Pavani Diwanji 在开发 Jeeves 项目时，将 Servlet 的概念重新启用，并发展 Servlet 成为一个具有实用性的技术。随后，Servelt API 伴随 Jeeves 项目结合到 Java Soft 的 Java Web Server。因此，Java Web Server 成为第一个支持 Servlet 技术的 Web 服务器。目前，许多 Web 服务器已经支持标准 Servlet API。目前，最新的 Servlet API 的版本是 2.5。

实质上，Servlet 程序是应用标准 Servlet API 的 Java 程序，这些程序可以扩展 Web 服务器的功能，实现强大的 Web 应用。Servlet 采用请求-响应 Web 模式。如图 13-1 所示，首先，用户向 Web 服务器提出请求。Web 服务器接受用户的请求，并定位所请求的 Servlet。然后，Servlet 根据不同发送的信息来处理请求，并动态生成 HTML 代码。最后，将生成的 HTML 代码响应在用户的浏览器，显示结果。

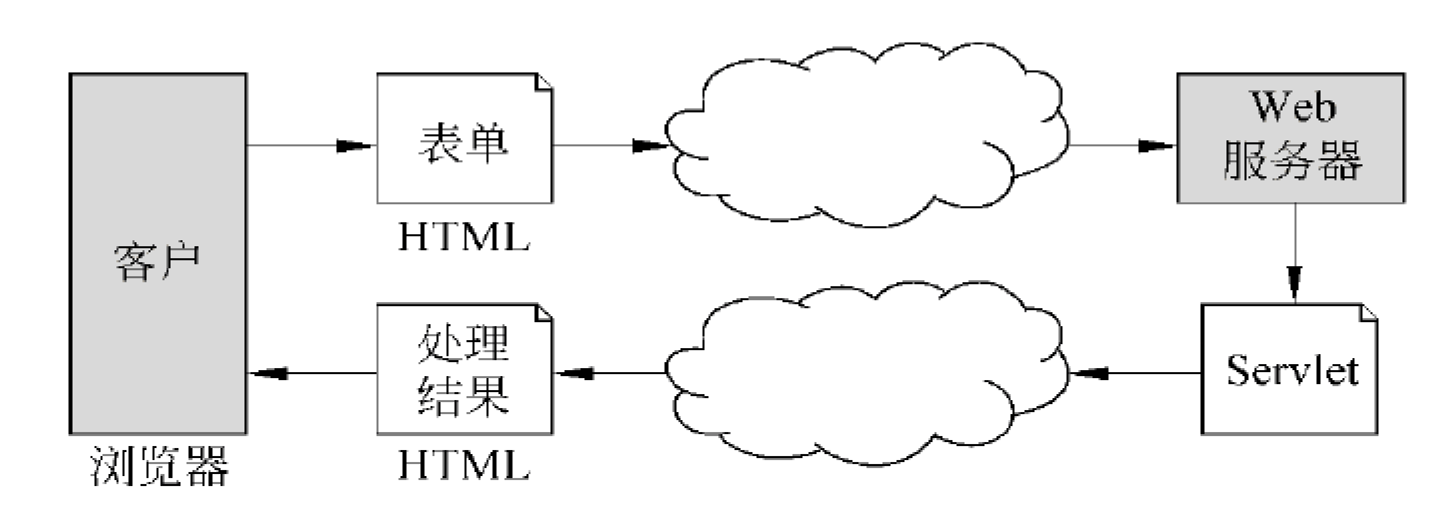


图 13-1　Servlet 的工作原理示意

从图 13-1 可以发现，Servlet 技术类似 CGI(公共网关接口)脚本，都是响应用户的请求，并生成动态的 HTML 代码，在客户端的浏览器中显示。但与 CGI 技术相比，Servlet 技术表现优于 CGI 技术的特点。

(1) Servlet 执行效率高。这是因为，CGI 在用户每次请求时，Web 服务器都要重新开始一个新进程。而 Servlet 启动由一个次要的 Java 线程实现，并常驻内存，实现了一次启动多次使用的作用。在执行效率上无疑优于 CGI。

(2) Servlet 继承了 Java 语言的特点，具有平台无关性。当前 Servlet API 规范完备，许多主流的 Web 服务器直接或间接支持 Servlet API。在此前提下，Servlet 可以跨平台实现，在不同的 Web 服务器中可以移植运行，无疑增加了 Servlet 的应用领域。

(3) Servlet 功能强大。Servlet 不仅可以动态生成 HTML 页面，更重要的是，用 Servlet 可以实现业务逻辑的处理与控制，如 Servlet 能够在各个程序之间共享数据，实现数据库连接池等功能。

（4）Servlet 学习简单，开发难度低。Sun 公司提供了 Servlet 教程和大量的范例，方便学习 Servlet 和 Servlet 的开发。

正因为 Servlet 具有这些特点，经过十多年的发展和完善，新技术层出不穷，Servlet 仍具有强大的生命力。当前，Servlet 主要应用在 JSP 体系的 Model 2 中作为控制组件，实现 MVC 的 Web 开发。从具体功能来说，Servlet 可以动态生成 HTML 页面，将信息广播到多个用户，可以与其他服务器中通信，如与数据库服务器交互，实现特殊复杂的应用。所以，Servlet 仍是学习的重点。

13.1.1　Servlet 的框架

Servlet API 是一个标准的 Java 扩展 API，主要包含了两个包：javax. servlet 和 javax. servlet. http。这两个包构成了 Servlet 的基本框架。javax. servlet 包主要定义了针对开发客户自定义协议应用的类和接口。而 javax. servlet. http 定义了大量的与应用于 HTTP 协议及相关的类和接口。

构成 Servlet 框架核心的接口和类有 javax. servlet. Servlet 接口、javax. servlet. GenericServlet 和 javax. servlet. http. HttpServlet。三者存在关联：javax. servlet. GenericServlet 实现了 javax. servlet. Servlet 接口，具有了 Servlet 接口的所有方法。而 javax. servlet. http. HttpServlet 是 javax. servlet. GenericServlet 的子类，也继承了 GenericServlet 的方法。

除此之外，javax. servlet. http 包中的 HttpServletRequest 接口和 HttpServletResponse 接口也是 Servlet API 的重要部分。它们为 Servlet 的请求—响应模式提供支持。因此，有必要对 HttpServletRequest 接口和 HttpServletResponse 接口有一个了解。

1．javax. servlet. Servlet 接口

javax. servlet. Servlet 接口是 Servlet 框架的基础。javax. servlet. Servlet 接口定义了所有 Servlet 必须实现的 5 个方法。

（1）void init(ServletConfig config)：由 Servlet 容器调用，初始化 servlet 对象。

（2）void service(ServletRequest req，ServletResponse res)：由 Servlet 容器调用，Servlet 容器会将用户请求信息封装在 ServletRequest 对象中，而将响应结果封装在 ServletResponse 对象中。

（3）void destroy()：是 Servlet 容器调用，表明 servlet 对象失效，释放占用空间。

（4）ServletConfig getServletConfig()：获取 servlet 的初始化信息以及参数信息。

（5）String getServletInfo()：返回 Servlet 的基本信息，如作者、版本和版权等。

init()、service()和 destroy()分别表示了一个 servlet 的生命周期的不同阶段，而 getServletConfig()与 getServletInfo()方法涉及 servlet 的相关信息。

实际上，开发 Servlet 类时，并不是直接实现 javax. servlet. Servlet 接口。而是通过扩展两个类来实现：javax. servlet. GenericServlet 和 javax. servlet. http. HttpServlet 类。

2．javax. servlet. GenericServlet 类

GenericServlet 类是一个抽象类，为开发独立协议应用的 Servlet 类提供支持。扩展 GenericServlet 类可以开发独立协议的 Servlet 类。对于继承 GenericServlet 类的 Servlet 子类必须实现 service()方法。

3. javax.servlet.http.HttpServlet 类

HttpServlet 是 GenericServlet 的子类，也是一个抽象类。扩展 HttpServlet 类，可以定义实现与 HTTP 协议相关的 Servlet 类，服务 Web 应用。

不同于 GenericServlet，HttpServlet 中实现了 service()方法，根据 HTTP 请求信息，调用相应的服务方法执行。这些方法有 doGet()、doPost()、doPut()、doDelete()、doOptions()、doTrace()等方法。假设从 service 方法的 HttpServletRequest 的对象获取的 HTTP 请求方式为 Get，则 service()方法会调用对应的 doGet()方法。

4. HttpServletRequest 接口

javax.servlet.http.HttpServletRequest 接口扩展 javax.servlet.ServletRequest，为 HTTP 请求提供信息。在具体执行过程中，Servlet 容器会将用户的请求信息封装在 HttpServletRequest 对象中，再将该 HttpServletRequest 对象传递给对应的服务方法，如 doGet()等。JSP 的内置对象 request 就是一个 HttpServletRequest 对象。

5. HttpServletResponse 接口

javax.servlet.http.HttpServletResponse 接口扩展 javax.servlet.ServletResponse，为发送 HTTP 响应结果提供服务。通常是 Servlet 容器创建 HttpServletResponse 响应对象，传递给对应的服务方法中，如 doGet()等。JSP 的内置对象 response 就是一个 HttpServletResponse 对象。

13.1.2 Servlet 的生命周期

Servlet 的生命周期表示从装载 Servlet 到 Servlet 终止之间的过程。Servlet 的生命周期分成 3 个阶段：初始化阶段、响应用户请求阶段和终止阶段，如图 13-2 所示。

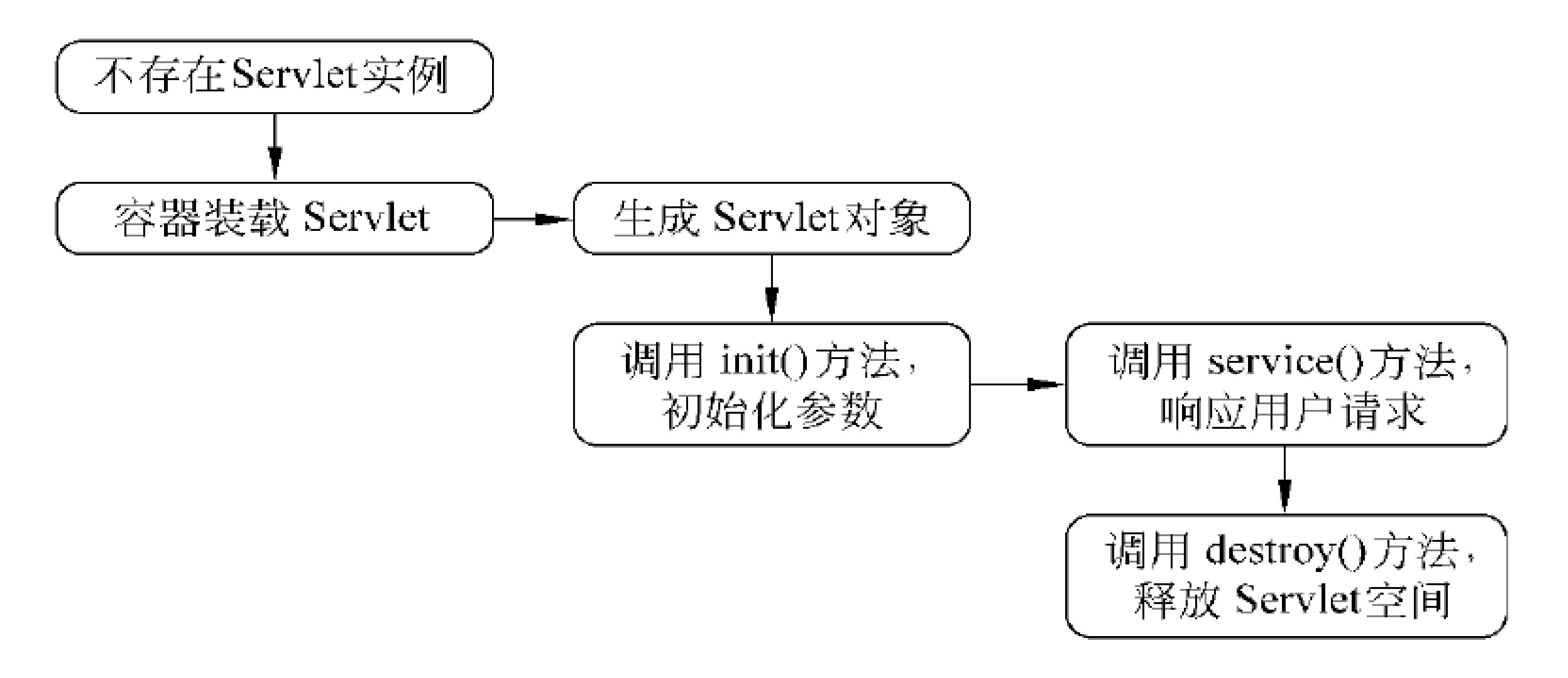


图 13-2 Servlet 的生命周期

1. 初始化阶段

初始化阶段中实际上包含了两个过程：装载 Servlet；初始化 Servlet。

通常，装载 Servlet 有以下几种情况。

(1) Servlet 容器启动时，根据配置自动装载 Servlet。一般，根据 web.xml 的<load-on-startup>属性，由该属性决定 Servlet 装载的顺序。

(2) Servlet 容器接收到用户的第一次请求。

(3) 由管理者决定。如果管理者修改了 Servlet 文件，Servlet 容器会重新装载 Servlet。

装载 Servlet 后，Servlet 容器会创建一个 Servlet 对象。并调用该 Servlet 对象的

init()方法，初始化 Servlet，分配资源，配置相关的参数。

2. 响应用户请求阶段

Servlet 容器会根据用户请求，生成 ServletRequest 对象和 ServletResponse 对象，分别封装用户请求的信息与响应请求结果。然后，调用 Servlet 对象的 service()方法，将生成的 ServletRequest 对象和 ServletResponse 对象，传递给 service()方法。在 service()中获取 ServletRequest 对象中用户信息，并进行处理，最后通过 ServletResponse 对象将响应发送给用户。

注意：如果用户的请求是基于 HTTP 协议的 Servlet，则 Servlet 容器会创建 HttpServletRequest 对象和 HttpServletResponse 对象来封装用户的请求和响应请求的结果。

3. 终止阶段

当 Servlet 被清除时，Servlet 容器会调用 Servlet 的 destroy()方法，释放 Servlet 占据的资源。

13.1.3 Servlet 的开发与部署

开发一个 Servlet 类，它必须是 javax. servlet. GenericServlet 的子类或 javax. servlet. http. HttpServlet 的子类。开发 Servlet 的最终目的是应用 Servlet，Servlet 不能直接被使用，必须在服务器中先部署 Serlvet。在本节中，将对开发 Servlet 的两种形式和部署 Servlet 进行介绍。

1. 开发 GenericServlet 的子类

一般，用 GenericServlet 子类来实现独立协议的客户应用。这个子类必须实现 GenericSevlet 类的抽象方法 service()。service 方法的声明形式如下：

```
public abstract void service (ServletRequest req,ServletResponse res)
                            throws ServletException,java.io.IOException
```

【例 13.1】 定义一个 GenericServlet 的子类 HelloGenericServlet，实现输出一个 html 网页，包含“Hello，Welcome into GenericServlet world”信息。具体代码见程序清单 13-1。

程序清单 13-1：

```
//HelloGenericServlet.java
package servlets;
import javax.servlet.* ;
import java.io.* ;
public class HelloGenericServlet extends GenericServlet{
    public void init(ServletConfig config) throws ServletException{                    //初始化
      super.init(config);
    }
public void service(ServletRequest req,ServletResponse res)
         throws ServletException,IOException{                    //响应请求
        res.setContentType("text/html;charset=gb2312");
          ServletOutputStream out=res.getOutputStream();
          out.println("<html>");
```

```
            out.println("<head>");
            out.println("<title>A Example of GenericServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("Hello,Welcome into GenericServlet World!");
            out.println("</body>");
            out.println("</html>");
        }
    public void destroy(){                                //释放资源
        }
    }
```

服务器端执行这个 Servlet，则会请求客户端的浏览器，通过创建的 ServletOutputStream 对象输出流 out 输出"Hello the new world"字符串。

2. 开发 HttpServlet 的子类

HttpServlet 子类定义 HTTP 协议相关的 Servlet。要定义一个 HttpServlet 子类，必须至少覆盖下列的方法之一。

(1) doGet(HttpServletRequest，HttpServletResponse)：如果 Servlet 支持 HTTP 的 Get 请求；方法声明格式如下：

```
protected void doGet(HttpServletRequest,HttpServletResponse)throws
ServletException,IOException
```

(2) doPost(HttpServletRequest，HttpServletResponse)：如果 Servlet 支持 HTTP 的 Post 请求；方法声明格式如下：

```
protected void doPost(HttpServletRequest,HttpServletResponse)throws
ServletException,IOException
```

(3) doPut(HttpServletRequest，HttpServletResponse)：如果 Servlet 支持 HTTP 的 Put 请求；方法声明格式如下：

```
protected void doPut(HttpServletRequest,HttpServletResponse)throws
ServletException,IOException
```

(4) doDelete(HttpServletRequest，HttpServletResponse)：如果 Servlet 支持 HTTP 的 Delete 请求；方法声明格式如下：

```
protected void doDelete(HttpServletRequest,HttpServletResponse)throws
ServletException,IOException
```

此外，也可以根据实际要求，对下列 HttpServlet 的常见方法定义。

(1) init()：执行生命周期中资源初始化。

(2) detroy()：执行生命周期中资源的释放。

(3) getServletInfo()：用于提供关于 servlet 自身信息。

【例 13.2】 定义一个 HttpServlet 的子类 HelloGenericServlet，可以输出一个包含"Welcome into HttpServlet World"信息的 html 网页。具体代码见程序清单 13-2。

程序清单 13-2：

```
//HelloHttpServlet.java
package servlets;
import java.io. * ;
import javax.servlet. * ;
import javax.servlet.http. * ;
public class HelloHttpServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException{
    response.setContentType("text/html;charset= gb2312");
    PrintWriter out= response.getWriter();
    out.println("< html> ");
    out.println("< head> ");
    out.println("< title> A Example of HttpServlet< /title> ");
    out.println("< /head> ");
    out.println("< body> ");
    out.println("Hello,Welcome into HttpServlet World!");
    out.println("< /body> ");
    out.println("< /html> ");
    }
    public void doPost(HttpServletRequest req,HttpServletResponse resp)
      throws ServletException,IOException{
    doGet(req,resp);
    }
}
```

上述的程序中定义 doGet()和 doPost()方法执行一样的功能，都是输出一个 html 文件。

当前的大部分 Web 应用是基于 Http 协议，并且 HttpServlet 子类定义形式简单，所以通常采用扩展 HttpServlet 来实现 Servlet 开发。

3. Tomcat 部署 Servlet

Tomcat 服务器中部署 Servlet 是通过设置 Web 应用对应的 WEB_INF\web. xml 文件实现。web. xml 配置 Servlet，即在 web. xml 文件中配置 Servlet 相关元素。Servlet 元素的子元素见表 13-1。

注意：为了部署 Servlet，先将编译 Servlet 的 class 文件保存在 Web 应用对应的 WEB-INF\classes 目录下。如果是指定包的类，需要将整个包保存在该目录下。如例 13-2 的 serlvets. HelloHttpServlet 类生成的 class 文件，保存的完整路径是“WEB-INF\classes\servlets\HelloHttpServlet. class”。

现在，用一个 web. xml 文件实现上述例子中对 HelloGenericServlet 和 HelloHttpServlet 的部署。具体代码见程序清单 13-3，执行结果如图 13-3 和图 13-4 所示。

表 13-1　Servlet 元素的常见子元素

元 素 名	说　　明
servlet-name	命名 Servlet 的名字
servlet-class	指定 Servlet 的类，不能与 jsp-file 元素同时使用
jsp-file	指定配置 JSP 文件作为 Servlet，不能与 servlet-class 元素同时使用
description	对 Servlet 的描述
load-on-startup	设置启动的优先级
init-param	初始化参数，参数的名称由子元素 param-name 指定，参数值由子元素 param-value 指定
servlet-mapping	表示 Servlet 的映射，允许用多个 servlet-mapping 表示为同一个 Servlet 做一个映射。用子元素 servlet-name 指定映射的 Servlet 的名字，用 url-pattern 子元素表示映射的 url 访问模式

程序清单 13-3：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--web.xml-->
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <description>Servlet 的应用实例</description>
  <servlet>  <!--部署 servlets.HelloGenericServlet-->
      <!--定义 servlet 名-->
      <servlet-name>HelloServlet1</servlet-name>
      <!--对应的 class-->
      <servlet-class>
          servlets.HelloGenericServlet
        </servlet-class>
  </servlet>
  <servlet-mapping>
      <servlet-name>HelloServlet1</servlet-name>
      <url-pattern>/Hello1</url-pattern>
  </servlet-mapping>
  <servlet>  <!--部署 servlets.HelloHttpServlet-->
      <servlet-name>HelloServlet2</servlet-name>
      <servlet-class>
        servlets.HelloHttpServlet
      </servlet-class>
  </servlet>
  <servlet-mapping>
      <servlet-name>HelloServlet2</servlet-name>
      <url-pattern>/Hello2</url-pattern>
  </servlet-mapping>
</web-app>
```



图 13-3 HelloGenericServlet 运行结果



图 13-4 HelloHttpServlet 运行结果

在 web.xml 中将上述定义在包 servlets 中的 HelloGenericServlet 和 HelloHttpServlet 分别命名为 HelloServlet1 和 HelloServlet2。将它们映射,对应的 url 模式分别为 Hello1 和 Hello2。在具体访问时,就可以在当前目录按照 url 模式对如图 13-3 和图 13-4 进行访问。

13.2 JSP 的开发模式

JSP 技术与 Servlet 技术都是基于 Java 语言的服务器端技术,已经成为开发 Web 应用的标准,被大量运用在 Web 应用的开发中。JSP 技术是在 Servlet 技术基础上发展起来的。所以,所有的 JSP 程序最终要转化成 HttpServlet 的子类才能运行。可以说,JSP 是 Servlet 的更高层次。JSP 的存在,可以方便开发大量的 HTML 形式的代码。

尽管如此,JSP 技术与 Servlet 技术仍存在着明显不同。

(1) JSP 技术的语法与 Servlet 不同,导致两者在编程方式上存在明显区别。Servlet 是用 Java 语言编写。而 JSP 是采用脚本语言的形式编写。这使得 JSP 代码中可以直接出现 HTML 标签,也可以用<%和%>符号将 Java 代码包含起来。

(2) Servlet 是用 Java 语言编写,任何 Servlet 程序必须先编译成类文件,才可以运行。对于 JSP 程序而言,不存在直接编译过程,它是由服务器的 JSP 容器来编译 JSP 成为 Servlet 类。因此,在第一次运行效率上存在区别,JSP 第一次运行速度慢于 Servlet。

(3) Servlet 作为 Web 服务器的组件,它可以是处理多种协议甚至自定义协议的

GenericServlet 子类，扩展服务器的功能。也可以是 HttpServlet 子类，实现基于 HTTP 的请求和响应。但是对于 JSP 而言，通过 JSP 容器编译后只能是 HttpServlet 的子类，处理基于 HTTP 协议的请求和响应。

在开发 Web 应用上，如何处理表现内容和业务逻辑，是采用 JSP 技术还是 Servlet 技术都是急需解决的问题。因此，根据 JSP 技术的特点以及 Web 应用的要求，Sun 公司推出了两种解决方案：JSP Model Ⅰ模式和 JSP Model Ⅱ模式。接下来，将了解这两种模式的不同与具体应用。

13.2.1　JSP Model Ⅰ：JSP＋JavaBean

JSP Model Ⅰ模式实质上是 JSP 与 JavaBean 相结合的技术，如图 13-5 所示。JSP 结合 JavaBean 有效地实现了视图与业务逻辑分离。

JSP Model Ⅰ模式实现 MVC(Model 模型-View 视图-Controller 控制器)模式。其中，JSP 实质上充当视图和控制器。用 JSP 页面独立接受用户的请求，以及处理响应请求，并将最终的响应结果返回给请求的客户。JavaBean 作为模型，实现数据访问和处理。

第 12 章的通讯录应用实例就是一个基于 JSP Model Ⅰ模式的 Web 应用。JSP 页面主要承担了视图和业务逻辑处理的功能。但是，这种模式存在一个问题，JSP 页面存在大量的 Java 代码，会导致程序逻辑关系不明确，难以阅读，增加了程序的复杂性和开发难度。在大型的 Web 应用开发上，会导致一些技术问题。因此，JSP Model Ⅰ模式往往应用在一些小型 Web 项目的开发上。

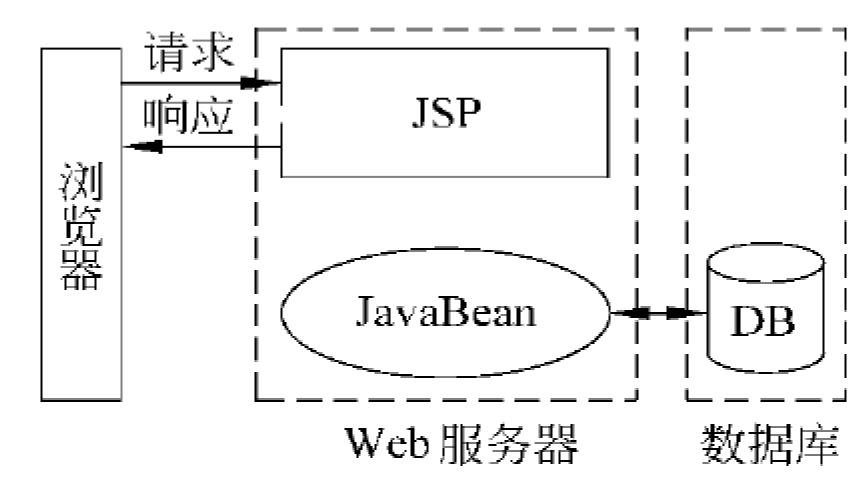


图 13-5　JSP Model Ⅰ模式

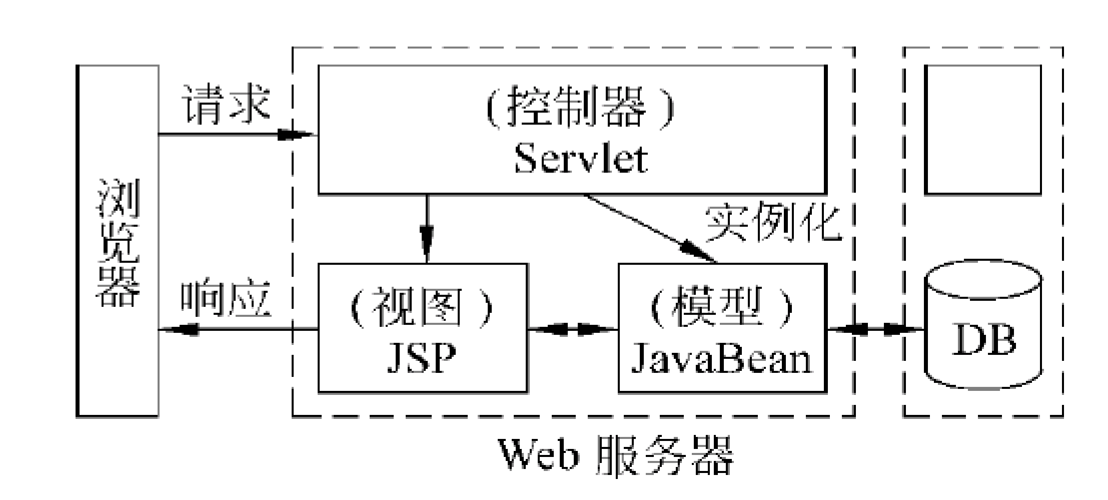


图 13-6　JSP Model Ⅱ模式

13.2.2　JSP Model Ⅱ：JSP＋JavaBean＋Servlet

JSP Model Ⅱ克服了 JSP Model Ⅰ存在的问题。JSP Model Ⅱ就是将 JSP＋JavaBean＋Servlet 技术结合起来的一种模式，如图 13-6 所示。JSP Model Ⅱ模式仍采用了 MVC 模式。

其中，一个或多个 Servlet 作为控制器。Servlet 接受请求，然后进行相应的业务逻辑处理后，创建并设置 JSP 页面的 JavaBean 对象，然后重新定向到 JSP 中。JavaBean 作为模型的角色，充当 JSP 和 Servlet 通信的中间工具，JavaBean 由 Servlet 创建，由 JSP 使用。JSP 作为视图，读取此 JavaBean 属性，然后进行显示。

与 JSP Model Ⅰ比较，可以发现，JSP Model Ⅱ模式结合 JSP 技术和 Servlet 技术的长处，一方面，Servlet 作为控制器的出现，充分体现了 Servlet 作为服务器端开发的优势，解决了 JSP 页面充斥大量的 Java 代码，页面难以开发和维护的局面。另一方面，JSP Model Ⅱ提高了 Web 应用开发的 MVC 设计。Servlet 的出现将 MVC 各层次的关系变得清晰明确，

改变了视图和控制器交错复杂的局面。JSP 技术也发挥了在表现内容上的优势，克服了在处理业务逻辑上的不足。通常，复杂的大型 Web 项目的开发会采用 JSP Model Ⅱ模式。具体的应用见 13.3 所示。

13.3 JSP+Servlet 的应用

作为 Web 服务器端的技术，JSP 和 Servlet 具有各自的特点。一般来说，JSP 对于动态生成的 HTML 页面内容具有优势，通常多被应用在网页内容的设计。对于 Servlet 技术而言，在表现业务逻辑控制处理方面，多被程序设计人员所擅长。结合 JSP 技术和 Servlet 技术，可以充分体现静态内容和动态内容的显示和 Web 业务逻辑的处理。在本节中将介绍 JSP 技术与 Servlet 技术的结合应用。

13.3.1 Servlet 实现会话管理

session 会话是在指定时间内服务器与用户之间的交互。通常用会话机制实现服务器对用户信息的跟踪。JSP 是通过内置对象 session 实现会话管理。JSP 的内置对象 session 是一个 JSP 容器创建的 javax.servlet.http.HttpSession 对象。对于 Servlet，必须在程序代码中创建一个 javax.servlet.http.HttpSession 接口的对象来实现会话管理。创建一个 javax.servlet.http.HttpSession 接口的对象依赖于封装用户请求的一个 javax.servlet.http.HttpServletRequest 对象。通过调用 HttpServletRequest 对象的 getSession()方法达到创建会话管理 HttpSession 对象的目的。

【例 13.3】 设计一个简易购物车应用。

本应用采用了 JSP Model Ⅱ模式。其中定义 Product.java 表示一个商品的 JavaBean，保存数据。shop.jsp 与 Cart.jsp 分别表示选择商品和选购商品的视图。ShoppingServlet 作为控制器，处理购物车中加入商品和删除商品。

(1) 定义选择商品界面。

程序清单 13-4：

```
<!--shop.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java" %>
<html>
<head>
<title>数码商店</title>
</head>

<body>
<p>欢迎进入数码商店</p>
<form name="shoppingForm" action="/shop/ShoppingServlet" method="POST">
  <b>移动硬盘</b>
  <select name="disk">
   <option>NewMan Mimi-Card(20G)|360</option>
   <option>DataTravler343(40G)|400</option>
```

```
    <option>FlyingWord44-d(60G)|500</option>
    <option>NewWorld MaxDisk(80G)|800</option>
   </select>
   <b>数量 </b>
   <input type="text" name="quantity" value="1" size="4">
   <input type="hidden" name="action" value="ADD">
   <input type="submit" name="Submit" value="加入购物车">
</form>
</body>
</html>
```

(2)购物车类商品的添加与删除。ShoppingServlet 定义一个购物车 cart,并将购物车添加到会话对象 session 中,实现在会话过程中的共享。通过 session 中 cart,实现将商品加入到购物车以及删除购物车中的商品功能。另外,程序中的重定向页面是通过一个 RequestDispatcher 的 Web 资源包装对象 view,重新包装请求与响应资源,通过 forward()方法实现页面的重定向。

程序清单 13-5:

```
//ShoppingServlet.java
package servlets;
import beans.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ShoppingServlet extends HttpServlet {
    public void doPost (HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {
    HttpSession session=req.getSession(false);      //创建一个 HttpSession 对象
    String action=req.getParameter("action");
    if(action==null)action="";                      //接受用户发送的动作请求
    RequestDispatcher  view;
    HashMap cart=(HashMap)session.getAttribute("shopping_cart");
                                                    //获取会话中 products 属性
    if (action.equals("ADD")){                      //加入购物车
        Product product=getProduct(req);            //实例化 Product 对象
        if(cart==null) {
            cart=new HashMap();
            cart.put(product.getName(), product);   //将 product 放入购物车
        }
        else{Product oldP=null;
            oldP=(Product)cart.get(product.getName());
            if(oldP!=null){
             product.setQuantity(oldP.getQuantity()+product.getQuantity());
            }
```

```
                cart.put(product.getName(),product);    //加入到购物车
                }
            session.setAttribute("shopping_cart", cart);            //将创建或修改后的购物车加入会话
            view= req.getRequestDispatcher("Cart.jsp");
               res.setContentType("text/html;charset= GB2312");
               view.forward(req, res);
        }
        else if(action.equals("DELETE")){                    //将商品从购物车中移除
        String quantityName= null;
        quantityName= req.getParameter("quantityName");
        if(quantityName! =null)
                cart.remove(quantityName);                   //删除商品
        view= req.getRequestDispatcher("Cart.jsp");
            res.setContentType("text/html;charset= GB 2312");
            view.forward(req, res);
        }
    }
    public void doGet (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
          doPost(req,res);
    }
    public Product getProduct(HttpServletRequest req)throws ServletException,IOException{
          String info= req.getParameter("disk");
            String quantityStr= req.getParameter("quantity");
            StringTokenizer t= new StringTokenizer(info,"|");
            String disk= t.nextToken();
            String priceStr= t.nextToken();
            int price= 0,quantity= 0;
            try{
               quantity= Integer.parseInt(quantityStr);
               price= Integer.parseInt(priceStr);
            }catch(NumberFormatException e){
                e.printStackTrace();
            }
            Product product= new Product();                //实例化 JavaBean
            product.setName(disk);
            product.setPrice(price);
            product.setQuantity(quantity);
            return product;
    }
}
```

(3) 选购商品到购物车,运行结果如图 13-7 所示。

程序清单 13-6:

```
<!--Cart.jsp-->
<%@ page contentType= "text/html; charset= gb2312" language= "java" import= "beans.Product,java.util.
```

```
*"%>
<html>
<head>
<title>数码商店</title>
</head>
<body>
  <jsp:include page="shop.jsp"/>
  <%
   HashMap cart=(HashMap)session.getAttribute("shopping_cart");
   if(cart!=null){                            //输出购物车的商品
   Set set=cart.entrySet();
   Iterator iterator=set.iterator();
   %>
   <table border="1" width="50%">
   <caption>购物车的商品</caption>
   <tr>
     <td>商品</td>
     <td>价格</td>
     <td>数量</td>
     <td> </td>
   </tr>
   <%
    while(iterator.hasNext()){                 //输出购物车的商品
     Map.Entry map=(Map.Entry)iterator.next();
     Product product=(Product)map.getValue();
     out.println("<tr><td>"+product.getName()+"</td>");
     out.println("<td>￥"+product.getPrice()+"元</td>");
     out.println("<td>"+ product.getQuantity()+"</td>");
      %>
     <form action="ShoppingServlet" method="POST">
     <td>
     <input type="hidden" name="quantityName" value="<%=product.getName()%>"/>
     <input type="hidden" name="action" value="DELETE"/>
     <input type="submit" value="删除"/>
      </td>
      </tr>
      </form>
     <%}%>
  </table>
  <%}
  %>
</body>
</html>
```

（4）商品信息组件。



图 13-7　购物车运行结果示意

程序清单 13-7：

```
//Product.java
package beans;
public class Product {
   private String name= "";
   private int price= 0;
   private int quantity= 0;
   public String getName() {
  return name;
   }
   public void setName(String name) {
  this.name= name;
   }
   public int getPrice() {
  return price;
   }
   public void setPrice(int price) {
  this.price=price;
   }
   public int getQuantity() {
  return quantity;
   }
   public void setQuantity(int quantity) {
  this.quantity= quantity;
  }
}
```

13.3.2 Servlet 实现文件操作

互联网为文件共享提供了方便。用户可以通过上传文件让别人使用,也可以通过下载文件使用。在实际操作中,为了网络安全,无论上传还是下载,Web 服务器中存放文件的目录往往对用户是不可见。所以,一般应用 Servlet 可以实现对文件的操作。

现在通过文件上传来了解 Servlet 对文件操作。实现文件上传,需要设计包含文件信息的表单。此时,表单中必须包含两个特性。

(1) 表单 form 的 method 属性必须设置为"post"。只有 post 值,才能发送文件,而 get 只能发送有限的信息,不能用在文件传输中。

(2) 必须设置表单的 enctype 属性为"multipart/form-data"。

设计一个文件上传的 JSP 页面 files.jsp,如程序清单 13-8 所示。

程序清单 13-8:

```
<!--files.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<title>文件上传</title>
</head>
<body>
<p>请选择上传文件</p>
<form  action="files/FileServlet" method="POST" enctype="multipart/form-data">
   <input type="file" name="upload" size="40"><br/>
   <input type="submit" value="提交">
</form>
</body>
</html>
```

实现文件上传,可以利用文件流,将文件的信息按字节依次读入到目标文件中。但是,这会导致目标文件前 4 行和后 5 行带有根据 HTTP 协议传递的表单信息。这使得目标文件与上传文件的原始内容存在差距。可通过 RandomAccessFile 实现对文件随机操作,截取前 4 行和后 5 行的内容,保留中间的有效内容。在下列程序清单 13-9 中,实现了文件的上传处理。

程序清单 13-9:

```
//FileServlet.java
package servlets;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FileServlet extends HttpServlet{
 String filePath;
 File path;
```

```
 File temp;
 File lastFile;
 public void init(ServletConfig config) throws ServletException{
      super.init(config);
      filePath=config.getInitParameter("uploadPath");   //从配置中获取保存目录名
      path=new File(filePath,"upload");                //创建文件目录
      path.mkdir();
      temp=new File(path.getParent()+"/"+path.getName(),"temp.tmp");
                                                       //建立临时文件
}
public void doPost(HttpServletRequest req,HttpServletResponse res)throws
      ServletException,IOException{
      InputStream in=req.getInputStream();
      FileOutputStream output=new FileOutputStream(temp);
      byte b[]=new byte[1000];                           //建立缓冲区
      int n;
      while((n=in.read(b))!=-1){                         //将文件写入到临时文件中
         output.write(b,0,n);
      }
      output.close();
      in.close();
      lastFile=new File(path.getParent(),getName(temp));
      PrintWriter out=res.getWriter();
      if(createFile())                                   //删除临时文件,创建保存文件
         out.println(lastFile.getName()+" Upload Success!Thank you");
      else
         out.println("Upload Fail");
}
public boolean createFile(){
       try{
       RandomAccessFile r1=new RandomAccessFile(temp,"r");
       RandomAccessFile r2=new RandomAccessFile(lastFile,"rw");
       long firstend=0,forth=1;
       int n;
       while((n=r1.readByte())!=-1&&(forth<=4)){        //定位第 5行
          if(n=='\n'){
               firstend=r1.getFilePointer();
               forth++;
          }
      }
      if(n==-1) return false;
      r1.seek(r1.length());
      long end=r1.getFilePointer();
      long mark=end;
      int j=1;
```

```
        while(mark>=0&&j<=6){                          //定位倒数第 6行
            mark--;
            r1.seek(mark);
            n=r1.readByte();
            if(n=='\n'){
                end=r1.getFilePointer();
                j++;
            }
        }
        r1.seek(firstend);
        long start=r1.getFilePointer();
        while(start<end-1){                            //将文件的有效内容读入到目标文件中
            n=r1.readByte();
            r2.write(n);
            start=r1.getFilePointer();
        }
        r2.close();
        r1.close();
        temp.delete();                                 //删除临时文件
        }catch(Exception e){
            e.printStackTrace();
        }
        return true;
    }
    public String getName(File file) {                 //获取临时文件内部包含的文件名
        String fileName=null;
        try{
        RandomAccessFile r=new RandomAccessFile(file,"r");
        int line=1;
        String fileInfo=null;
        while(line<=2){
         fileInfo=r.readLine();
         line++;
        }
         int loc=fileInfo.lastIndexOf("\\");
         fileName=fileInfo.substring(loc+1,fileInfo.length()-1);
         r.close();
         }catch(Exception e){
             e.printStackTrace();
         }
         return fileName;
      }
    }
```

上述实现文件上传方法复杂,也可以运用第 10 章介绍的 jspSmartUpload 组件完成文件上传。请自行编写代码。Servlet 技术也可以实现文件下载。文件下载的原理类似于文

件上传。Servlet 利用 java.io 包的输入输出流，用文件流依次读取复制链接的文件，达到下载文件的目的。在这里就不详细介绍了，请读者自行完成。

13.3.3 Servlet 实现数据库的访问

数据库是实现 Web 应用的重要内容。Servlet 技术同 JSP 技术一样，结合 JDBC 技术实现对数据库的连接，访问数据库，达到生成动态内容的目的。在 JSP Model Ⅱ中，可以通过 Servlet 技术访问数据库，达到实例化 JavaBean 对象的目的。

本节将通过一个用户登记注册的应用来了解 Servlet 实现数据库的访问。同时，进一步了解用 JSP Model Ⅱ模式开发 Web 应用。充分体会 JSP、JavaBean 和 Servlet 在应用中实现 MVC 模式。

【例 13.4】 利用 Servlet 实现用户登录、注册控制的 Web 应用。

该应用的目的是，用户通过进入登录 JSP 页面，在登录页面中选择登录还是注册。根据用户动作的选择，发送信息到控制器 Servlet 中，由 Servlet 处理不同的动作请求，并给予处理，根据处理结果不同，重新得到不同的 JSP 页面。涉及的文件清单见表 13-2，具体的代码见程序清单，运行结果如图 13-8～13-11 所示。

表 13-2 用户登记注册应用的文件清单

类　别	文件名	描　　述
JSP	index.jsp	定义网站的首页，允许用户选择登录或注册
	register.jsp	用户填写注册信息
	error1.jsp	用户登录失败
	error2.jsp	用户注册失败
	welcome.jsp	用户成功登录后的欢迎页面
JavaBean	UserBean.java	代表用户
Servlet	UserServlet	定义控制器，处理用户发出的所有动作
数据库	user 数据库目录下的 userlist.frm	定义保存用户信息的数据表 userlist，userlist 包含 3 个文本字段(email，username，password)，分别表示 email 账号、昵称和登录密码

程序清单 13-10：

```
<!--index.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java" %>
<html>
<head>
<title>主页</title>
</head>
<body>
<p align="center">
  <form action="/user/UserServlet" method="post">
   <table border="1">
     <caption>用户登录</caption>
```

```
        <tr>
          <td>email</td>
          <td><input type="text" name="email"/>
        </tr>
        <tr>
          <td>password</td>
          <td><input type="password" name="password"/>
        </tr>
        <tr>
            <td></td>
            <td>
                <input type="submit" name="action" value="login"/>
                <input type="submit" name="action" value="register"/>
      </table>
    </form>
</p>
</body>
</html>
```

程序清单 13-11：

```java
//UserBean.java
package beans;
public class UserBean {
    private String username;                    //昵称
    private String email;                       //email
    private String password;                    //密码
    public String getEmail() {
      return email;
    }
    public void setEmail(String email) {
      this.email=email;
    }
    public String getPassword() {
      return password;
    }
    public void setPassword(String password) {
      this.password=password;
   }
    public String getUsername() {
      return username;
    }
    public void setUsername(String username) {
      this.username=username;
    }
}
```

程序清单 13-12：

```
<!--register.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java" %>
<html>
<head>
<title>主页</title>
</head>
<body>
<p align="center">
  <form action="/user/UserServlet" method="post">
   <table border="1">
     <caption>用户注册</caption>
     <tr>
      <td>email</td>
      <td><input type="text" name="email"/>
     </tr>
     <tr>
      <td>昵称</td>
      <td><input type="text" name="username"/>
     </tr>
     <tr>
      <td>password</td>
      <td><input type="password" name="password"/>
     </tr>
     <tr>
       <td></td>
       <td>
          <input type="submit" name="action" value="confirm"/>
          <input type="reset" value="clear"/>
   </table>
  </form>
</p>
</body>
</html>
```

程序清单 13-13：

```
//UserServlet.java
package servlets;
import beans.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
public class UserServlet extends HttpServlet{
      public void doGet(HttpServletRequest request,HttpServletResponse
```

```
response)throws
        ServletException,IOException{
    String action=request.getParameter("action");
    String email=request.getParameter("email");
    String password=request.getParameter("password");
    String username=request.getParameter("username");
    if(action==null)action="";
    if(email==null)email="";
    if(password==null)password="";
    if(username==null)username="";
    UserBean user=new UserBean();
    RequestDispatcher  view;
    //处理登录请求动作
    if(action.equals("login")){                     //登录动作
    user=queryARecord(email,password);
    if(user! =null){                                //登录成功
        HttpSession session=request.getSession();
        session.setAttribute("userinfo", user);
    view=request.getRequestDispatcher("welcome.jsp");
    view.forward(request, response);                //重定向到 welcome.jsp
    }
    else{                                           //登录失败
     view=request.getRequestDispatcher("error1.jsp");
     response.setContentType("text/html;charset=GB 2312");
         view.forward(request, response);           //重定向到 error1.jsp
    }
    }
    //处理注册请求动作
    else if(action.equals("register")){             //转向 register.jsp
    view=request.getRequestDispatcher("register.jsp");
        response.setContentType("text/html;charset=GB 2312");
    view.forward(request, response);
    }
    //处理登录信息插入到数据表中
    else if(action.equals("confirm")){
    user.setEmail(email);                           //实例化 JavaBean 的 user 对象
    user.setUsername(username);
    user.setPassword(password);
    if(insertARecord(user)){                        //插入到数据表中
            HttpSession session=request.getSession();
            session.setAttribute("userinfo", user);
                                                              //设置 session 属性 userinfo
            view=request.getRequestDispatcher("welcome.jsp");
            view.forward(request, response);        //重定向到 welcome.jsp
          }
          else{
            view=request.getRequestDispatcher("error2.jsp");
```

```
                view.forward(request, response);              //重定向到 error1.jsp
            }
    }
}
public void doPost(HttpServletRequest request,HttpServletResponse response)throws
    ServletException,IOException{
        doGet(request,response);
}
    public Connection getConnection() {
        Connection connection=null;
        String driver="com.mysql.Jdbc.Driver";              //默认驱动程序
        String jdbcurl="mysql://localhost=3306/user";       //jdbcurl 数据源
        try{
                Class.forName(driver);                      //注册驱动程序
                connection=DriverManager.getConnection(jdbcurl,"root","");                //建立连接
            }catch(ClassNotFoundException e1){
                e1.printStackTrace();
            }catch(SQLException e2){
                e2.printStackTrace();
            }
        return connection;
}
public void closeConnection(Connection connection){           //关闭连接
        try{
            if(connection!=null)
                connection.close();
            connection=null;
        }catch(SQLException e3){
            e3.printStackTrace();
        }
}
public void closePstmt(PreparedStatement pstmt){              //关闭执行语句
        try{
            if(pstmt!=null)
                pstmt.close();
            pstmt=null;
        }catch(SQLException e){
            e.printStackTrace();
        }
}
public void closeResultSet(ResultSet rs){                     //关闭结果集语句
        try{
            if(rs!=null)
                rs.close();
```

```
            rs=null;
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
    public UserBean queryARecord(String email,String password){                    //查询记录
        String queryStr="select * from userlist where email=?and password=?";
        Connection conn=null;
        PreparedStatement pstmt=null;
        ResultSet rs=null;
        UserBean result=null;
        if(email==null||password==null) return result;
        try{
        conn=getConnection();
        pstmt=conn.prepareStatement(queryStr);
        pstmt.setString(1, email);
        pstmt.setString(2, password);
        rs=pstmt.executeQuery();
        if(rs.next()) {
            result=new UserBean();
            result.setEmail(rs.getString(1));
            result.setUsername(rs.getString(2));
            result.setPassword(rs.getString(3));
        }
        }
        catch(SQLException e){
            e.printStackTrace();
        }
        finally{
          closeResultSet(rs);
          closePstmt(pstmt);
          closeConnection(conn);
        }
        return result;
    }
    public boolean insertARecord(UserBean user){                //插入记录
       String insertStr="insert into userlist values (?,?,?)";
        Connection conn=null;
        PreparedStatement pstmt=null;
        ResultSet rs=null;
        if(user==null) return false;
        try{
        conn=getConnection();
        pstmt=conn.prepareStatement(insertStr);
        pstmt.setString(1, user.getEmail());
```

```
            pstmt.setString(2,user.getUsername());
            pstmt.setString(3, user.getPassword());
            if(pstmt.executeUpdate()!=0) return true;
            }catch(SQLException e){
                e.printStackTrace();
                return false;
            }finally{
              closeResultSet(rs);
              closePstmt(pstmt);
              closeConnection(conn);
            }
            return false;
    }
}
```



图 13-8 登录界面运行结果

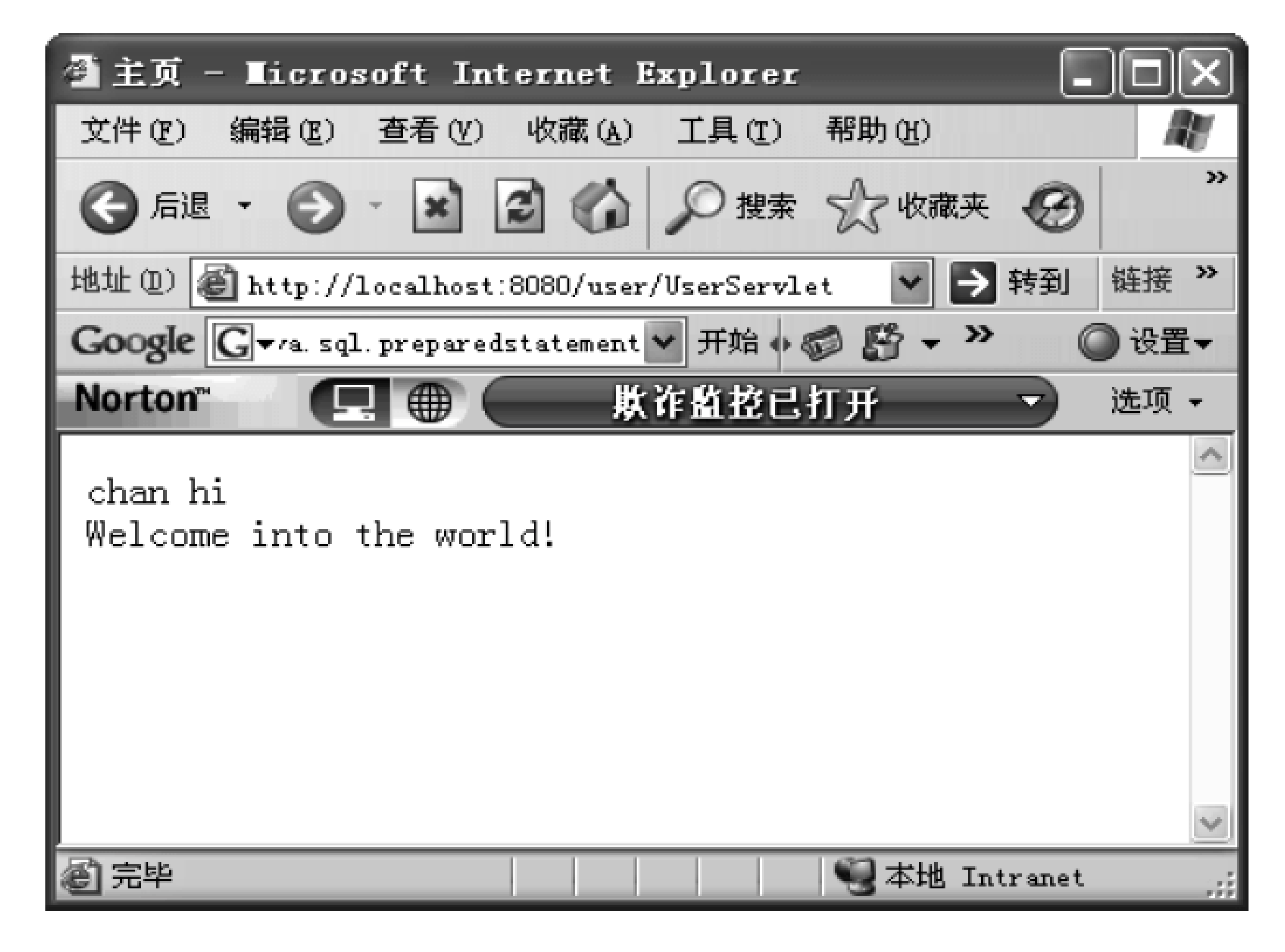


图 13-9 登录成功界面

图 13-10　登录失败界面

图 13-11　注册界面 register.jsp

UserServlet.java 定义了一个控制器 Servlet，可以根据用户动作控制处理。可以处理与执行 index.jsp 发送的登录(login)、注册请求(register)以及 register.jsp 的插入用户注册信息的动作(confirm)控制。根据用户的请求动作(请求登录处理、请求注册处理、请求插入注册信息)，执行相应数据库连接和访问(查询数据、插入数据)，执行业务，然后重定向页面。

程序清单 13-14：

```
<!--welcome.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"
import="beans.UserBean"%>
<html>
<head>
<title>主页</title>
</head>
```

```
<body>
  <%
   UserBean user= (UserBean)session.getAttribute("userinfo");
   if(user! =null) out.println(user.getUsername()+"<br>");
   %>
  Welcome into the world!
</body>
</html>
```

程序清单 13-15：

```
<!--error1.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>登录错误信息</title>
</head>
<body>
  错误的账号与密码,重新登录!
  <jsp:include page="index.jsp"/>
</body>
</html>
```

程序清单 13-16：

```
<!--error2.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<html>
<head>
<title>主页</title>
</head>
<body>
  已有账号,请重新注册!
  <jsp:include page="register.jsp"/>
</body>
</html>
```

小　　结

本章中从 Servlet 的基本概念出发,依次介绍 Servlet 的工作原理、Servlet 的框架和常见的类和接口,以及 Servlet 的生命周期。此外,根据应用不同要求,介绍了独立协议的 GenericServlet 的子类的定义,以及实现 HTTP 协议的 HttpServlet 的定义。详细说明了 Servelt 的开发与在 Tomcat 服务器上的部署。比较了 JSP 的两种体系架构,阐述了 JSP Model Ⅰ和 JSP Model Ⅱ模式的特点和应用领域。此外,通过具体实例介绍了 Servlet 技术

结合 JSP 技术的常用应用，来强化 Servlet 技术的概念的理解和在 Web 开发中的角色。

练　习　13

1．填空题

（1）Servlet 是________。

（2）Servlet API 包含了________和________包。

（3）Servlet 的生命周期分成________、________和________这 3 个阶段。

（4）JSP Model Ⅱ工作模式是结合________、________和________技术。

（5）Servlet 中通过________来创建 HttpSession 对象。

2．选择题

（1）下列说法正确的是________。

A．javax.servlet.Servlet 是一个接口

B．HttpServeltRequest 接口提供响应用户请求的方法

C．HttpServletRequest 接口封装用户请求信息

D．Servlet 容器创建 HttpServletRequest 对象来封装用户请求信息

（2）下列代码存在错误的是________。

A．第 4 行　　B．第 8 行　　C．第 11 行　　D．第 12 行

```
第 1行： package servlets;
第 2行： import javax.servlet.* ;
第 3行： import java.io.* ;
第 4行： public class HelloHttpServlet extends HttpServlet{
第 5行： public void init(ServletConfig config) throws ServletException{
第 6行： super.init(config);
第 7行： }
第 8行： public void service(ServletRequest req,ServletResponse res) throws ServletException,
IOException{
第 9行： res.setContentType("text/html;charset=gb2312");
第 10行： ServletOutputStream out=res.getOutputStream();
第 11行： out.println("Hello,Welcome into GenericServlet World!");
第 12行： }
第 13行： public void destroy(){
第 14行： }
第 15行： }
```

3．实验题

设计一个采用 JSP Model Ⅱ模式的留言板。要求可以创建论题、回复、浏览论题和回复。

第 14 章　JSP 和 XML

14.1　JSP 生成 XML

XML 是具有严格的语法规则的良构性标记语言。在第 5 章主要讨论了 XML 技术以及相关技术在客户端的应用。XML 实质上是服务器端技术，本章将从 XML 在服务器端的应用展开介绍。当前许多 Web 应用都是基于 XML 技术的。这是因为，与数据库相比，XML 具有编写简单、跨平台、不需要特别的数据库管理软件，就可以保存海量的信息。XML 数据间关系简单明了，不需要范化复杂的关系模式。更重要的是，XML 技术得到广泛的支持。大部分软件开发工具可以方便地访问 XML 文件，获取数据，实现数据的共享，利用 XML 数据可以动态生成各种形式的文件。在当前数据驱动 Web 应用的前提下，XML 数据成为 Web 应用的核心内容，XML 往往与其他 Web 技术（如 JSP、ASP.net 等）结合，建立功能强大的各种 Web 服务。

JSP 技术是开发基于 XML 技术的 Web 应用的主流技术之一。作为 Java 软件开发环境的一部分，JSP 利用 Java API 处理 XML 数据，具体涉及 JSP 实现对生成各种形式基于 XML 文件如 HTML、XHTML、WML 等，XML 文件的解析、JSP 应用 XML 和转换 XML 文件。

本节中将讨论 JSP 生成 XML。JSP 可以应用 Java API 动态生成 XML、HTML 和其他形式的页面。但是这种方式会使 JSP 页面内嵌入大量的 Java 语言脚本，导致 JSP 文件内容复杂。JSP 技术中提供了非常简单的方式动态实现各种基于 XML 的文件，主要有两种方式：

（1）JSP 直接生成 XML 文件；

（2）JSP 结合 JavaBean 生成 XML 文件。

14.1.1　JSP 直接生成 XML 文件

JSP 中要显示 XML 文件，只需要在 JSP 文件用设置指令<%@ page%>的 contentType 属性为“text/xml”即可。下列的程序清单 14-1，显示了用 JSP 直接生成 XML 文件的应用。运行结果如图 14-1 所示。

程序清单 14-1：

```
<!--JSP14-1.jsp-->
<%@page contentType="text/xml"%>
<?xml version="1.0" encoding="UTF-8"?>
<studentlist>
    <student>
        <id>90123</id>
        <name>黄海</name>
        <birthday>1990-12-10</birthday>
        <gendar>男</gendar>
    </student>
```

```
    <student>
        <id>90124</id>
        <name>刘沙</name>
        <birthday>1992-02-19</birthday>
        <gendar>女</gendar>
</student>
</studentlist>
```

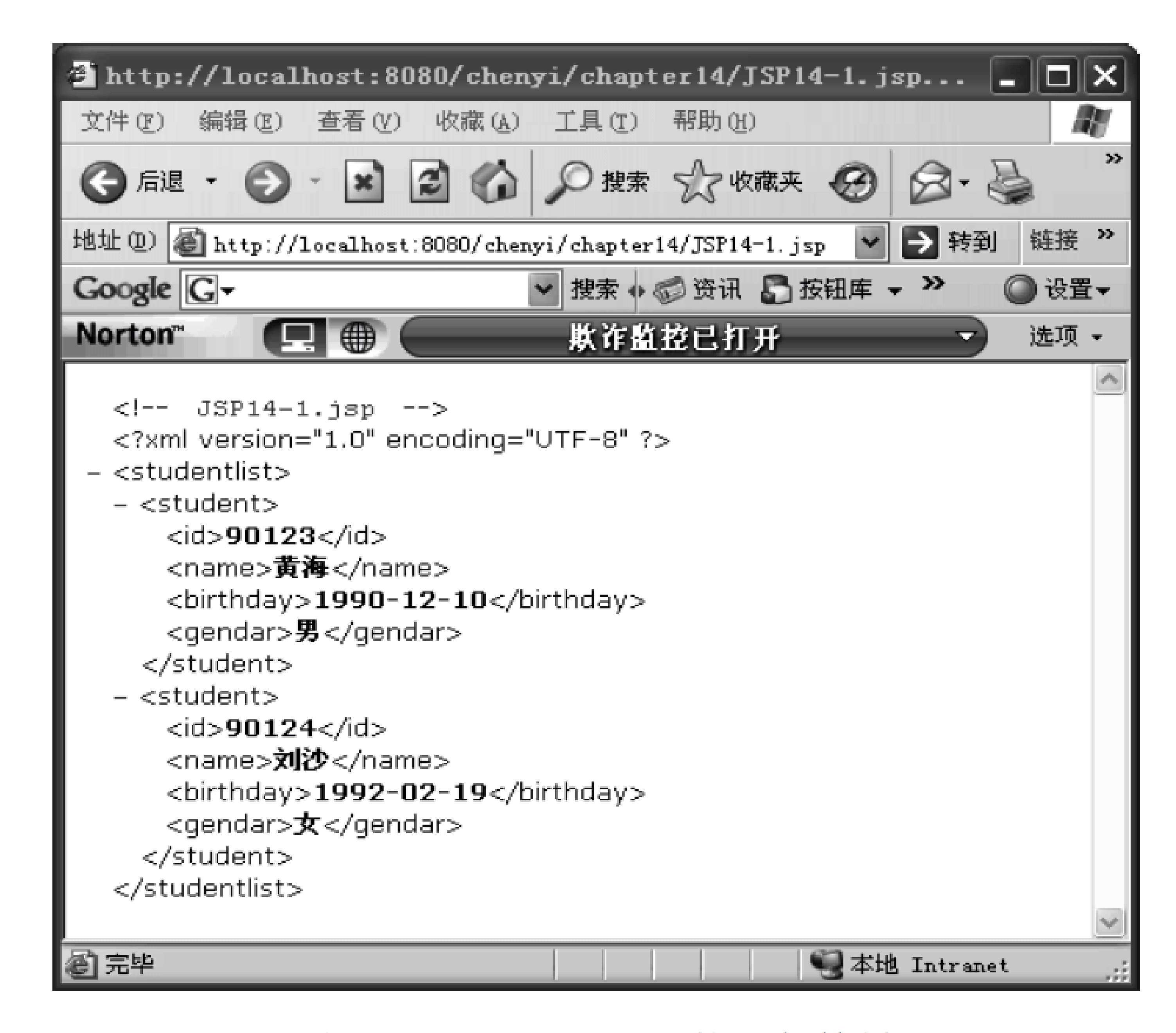


图 14-1　JSP14-1.jsp 的运行结果

任何基于 XML 的其他格式文件也可以通过类似方式直接生成。假设为了直接生成 WML 文件，可在 JSP 页面中直接用<%@ page contentType="text/vnd.wap.wml"%>进行定义，生成的文件就是 wml 文件形式。程序清单 14-2 展示了用 JSP 生成一个 WML 内容的实例，运行结果如图 14-2 所示。

程序清单 14-2：

```
<!--JSP14-2.jsp-->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD WML 2.0//EN"
"http://www.wapforum.org/dtd/wml20.dtd">
<%@page contentType="text/vnd.wap.wml"%>
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:wml="http://www.wapforum.org/2001/wml">
<head>
    <title>Demo</title>
</head>
    <wml:card id="first" title="Menu">
```

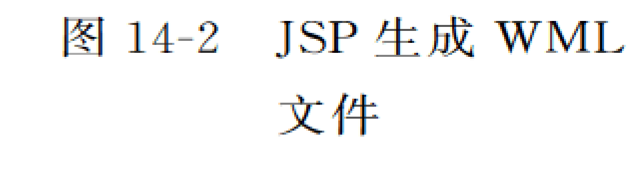


图 14-2　JSP 生成 WML 文件

```
        <p>
        Welcome into <b>JSP</b> and <b>XML</b> World.
        </p>
    </wml:card>
</html>
```

14.1.2 结合 JavaBean 生成 XML 文件

JSP 利用 JavaBean 组件也可以生成基于 XML 技术的各种文件，包括 XML 文件本身。通过这种方式，JSP 充分利用了 JavaBean 组件对象实现开发多种不同终端，类似 Web 应用的需要。这在具体应用 JavaBean 对象生成 XML 文件时，先必须将 JavaBean 类导入到 JSP 文件中，再在 JSP 页面中用<jsp:useBean>实现创建 JavaBean 对象，达到使用该组件数据的目的。

程序清单 14-3 是将 12.3.3 小节中的程序清单 12-16 改写而成，具体功能是输出通讯录的所有记录。

程序清单 14-3：

```
<!--JSP14-3.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="beans.*,java.sql.*" errorPage="error.jsp"%>
<jsp:useBean id="conn" scope="session" class="beans.AddressDBean"/>
<?xml version="1.0" encoding="UTF-8"?>
<AddressList>
   <%AddressBean[] records=conn.getAllRecords();
      if(records!=null){
         for(int i=0;i<records.length;i++){
      %>
       <Record>
         <Address_Id><%=records[i].getAddr_Id()%></Address_Id>
         <Address_Name><%=records[i].getName()%></Address_Name>
         <Phone><%=records[i].getPhone()%></Phone>
         <Address><%=records[i].getAddress()%></Address>
         <Email><%=records[i].getEmail()%></Email>
      </Record>
      <%
         }
       }
     %>
</AddressList>
```

如果需要生成的 XML 文件（如图 14-3 所示）按照一定格式进行显示，如图 14-4 所示，可以在<? xml? >后面声明转换的 XSLT 文件，如“<? xml-stylesheet type = "text/xsl" href="addresslist.xslt"? >”，用 addresslist.xslt 文件来转换生成的 XML 文件为指定格式的 HTML 页面。也可用“<? xml-stylesheet type="text/css" href="css 文

件"? >"指定一个 CSS 文件,来设置 XML 文件的显示格式。

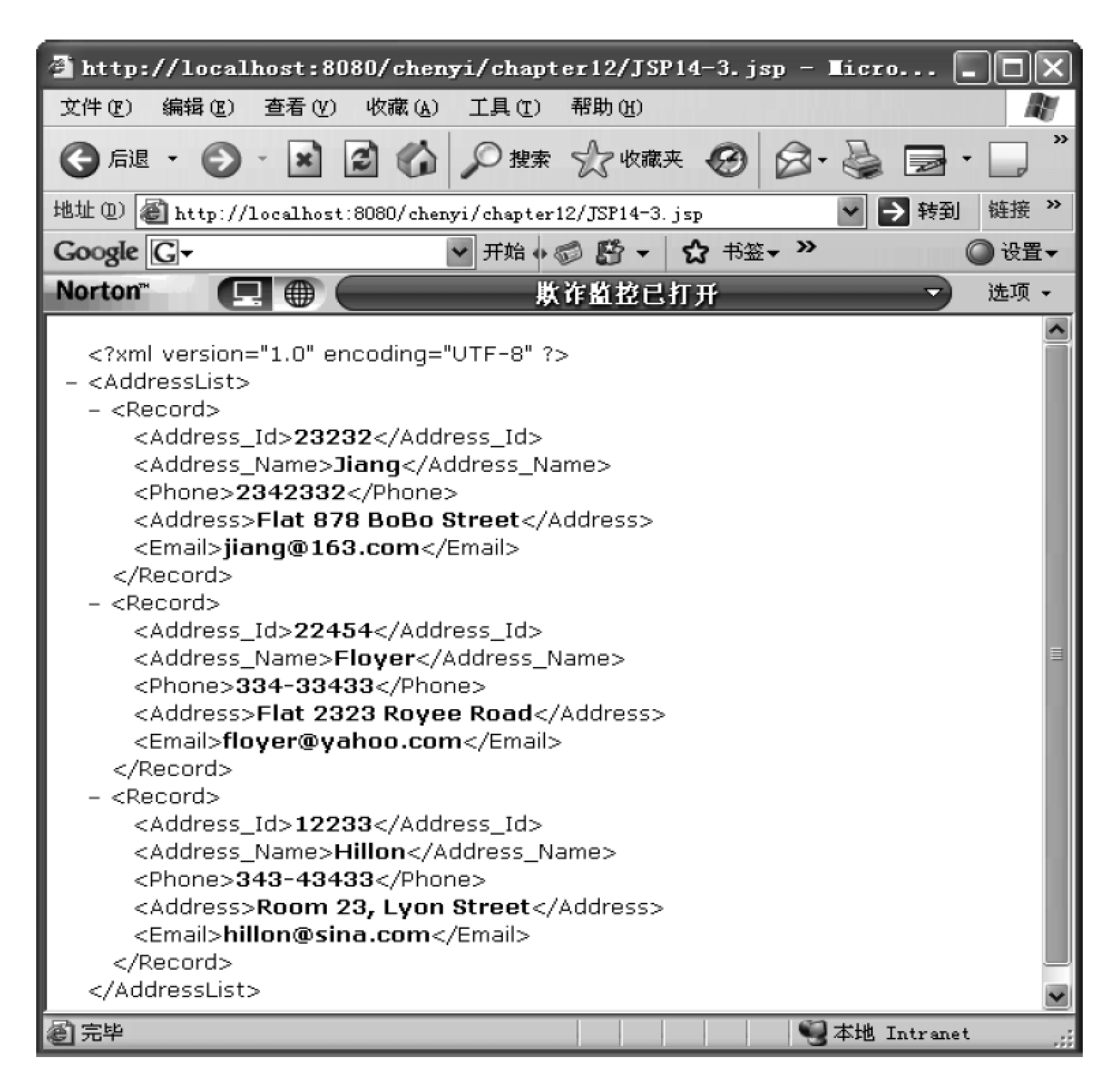


图 14-3　JSP 生成的原始 XML 文件



编号	姓名	电话	通信地址	Email
23232	Jiang	2342332	Flat 878 BoBo Street	jiang@163.com
22454	Floyer	334-33433	Flat 2323 Royee Road	floyer@yahoo.com
12233	Hillon	343-43433	Room 23, Lyon Street	hillon@sina.com

图 14-4　JSP 生成 XSLT 转换的 XML 文件

14.2　JSP 解析 XML

JSP 开发 XML 的 Web 应用,解析 XML 文件是一个重要前提。通过解析器解析 XML 数据,将它们分成开始标签、标签属性、标签主体、结束标签不同部分,达到理解 XML 文件的目的,做出相应的处理。

当前可以解析 XML 数据有两种形式：DOM(Document Object Model 文件对象模型)和 SAX(Simple API for XML，支持 XML 简单应用接口)。Sun 公司开发了 JAXP API 提供了支持解析 XML 数据的这两种形式。

14.2.1　JAXP API 概述

JAXP API(API for Java XML Processing) 是 Sun 公司研发用于处理 XML 的应用接口。JAXP API 提供了类和接口，实现 JAVA 程序或 JSP 进行 XML 数据解析、转换、验证以及 XML 数据的查询。JAXP API 为开发实现 XML 的应用提供了技术支持。

从 J2SE 1.4 中实现 JAXP API 1.1 以来，JAXP API 已经成为 Java 平台的标准组件，不需要另外安装。JAXP API 伴随技术更新发生不断变化，版本不断更新。J2SE 5.0 支持 JAXP 1.3。当前主流的 J2SE 6.0 版本已经支持 JAXP API 1.4。从单纯 JAXP API 1.0 支持 SAX 1.0 和 DOM 的 XML 解析，到目前 JAXP API 实现对 XML 多种形式的处理，JAXP API 已经成为开发 Java/XML 应用的重要部分。

JAXP API 1.4 是对当前的主流 XML 及相关技术提供标准接口，具体包括了 XML 1.0、XML 1.1、XSLT 1.0、XPath 1.0、SAX 2.02、DOM Level 3 等内容。此外，还增加了支持 XML 的流应用接口 STAX API(Streaming API for XML)。具体内容见表 14-1。

表 14-1　JAXP API 1.4 常见的包

包　　名	说　　明
javax. xml. datatype	XML/Java 类型的映射
javax. xml. namespace	XML 命名空间的处理
javax. xml. parsers	处理 XML 文档，支持两种可插入解析器 SAX 和 DOM
javax. xml. transform	定义了用于处理转换指令
javax. xml. transform. dom	提供特定 DOM 转换的应用接口
javax. xml. transform. sax	提供特定 SAX 2.0 转换的应用接口
javax. xml. transform. stream	实现特定于流和 URI 的转换 API
javax. xml. validation	提供 XML 文档验证的应用接口
javax. xml. xpath	此包提供了用于 XPath 表达式的计算和访问计算环境的对象模型的中立接口
org. w3c. dom	为文档对象模型 DOM 提供支持
org. xml. sax	支持核心 SAX 2.0 API
org. xml. sax. ext	提供 SAX 2.0 扩展处理
org. xml. sax. helpers	提供解析错误处理的相关类和接口

不管 JAXP API 的功能发生怎样的变化，有一点可以肯定的是：对 XML 数据输入处理是 JAXP API 的一项重要功能，也是实现其他功能的前提。JAXP API 提供了应用接口 DOM API 实现 XML DOM 和 SAX API 实现 SAX 2.0.2 的 XML 文件输入处理以及解析。在下两节中，将对 JSP 具体实现两种解析 XML 文件方式展开介绍。

14.2.2 JSP 应用 DOM

DOM(Document Object Model)是 W3C 推荐标准,是文档结构化表示、访问以及操作的应用程序接口,与平台和语言无关。它提供实现程序和脚本动态地访问修改文件的内容、结构和样式的方法。

W3C DOM 从发展到成熟经历了 3 个主要阶段。1998 年 W3C 推出 DOM Level 1,是 DOM 的核心层,针对 HTML 和 XML 文档模型,实现对 HTML 和 XML 文档的导航以及相关处理。伴随着技术进步,W3C 与 2000 年推出 DOM Level 2,在 DOM Level 1 的基础上增加样式表对象模型、事件模型以及提供了对 XML 命名空间的支持。目前使 W3C 推荐标准是 DOM Level 3。DOM Level 3 在 DOM Level 2 的基础上,又具有了内容模型和文档验证、文档的加载保存、文档的查看、格式化和关键事件。

JAXP API 1.4 提供了 DOM API,支持 DOM Level 3 的核心层(DOM Level 3 Core)以及加载保存层(DOM Level 3 Load and Save),见表 14-1。DOM API 的 org.w3c.dom 包提供了对 XML 文档的所有元素和属性处理的接口。如表 14-2 所示,通过它们可以实现对 DOM 树的遍历、修改等。这使得基于 Java 语言的程序和脚本利用 JAXP API 实现 XML 处理。

表 14-2 org.w3c.dom 包的常见接口

接　　口	说　　明
Document	具有树状结构的 XML 文件或 HTML 文件
Node	结点,是 DOM 的基本数据
NodeList	结点集合
Element	XML 文件或 HTML 文件的元素
Attr	元素对象的属性
Text	元素和属性的文本内容

JSP 应用 JAXP API 实现对 XML 的输入处理。具体解析处理过程是:DOM API 加载 XML 文档到内存,解析器解析 XML 文档,将 XML 文档的各种成分处理为结点,这些结点构成一个结点树。然后,在这个结点树中利用结点之间的关系实现遍历和执行插入删除等相关的处理,如图 14-5 所示。

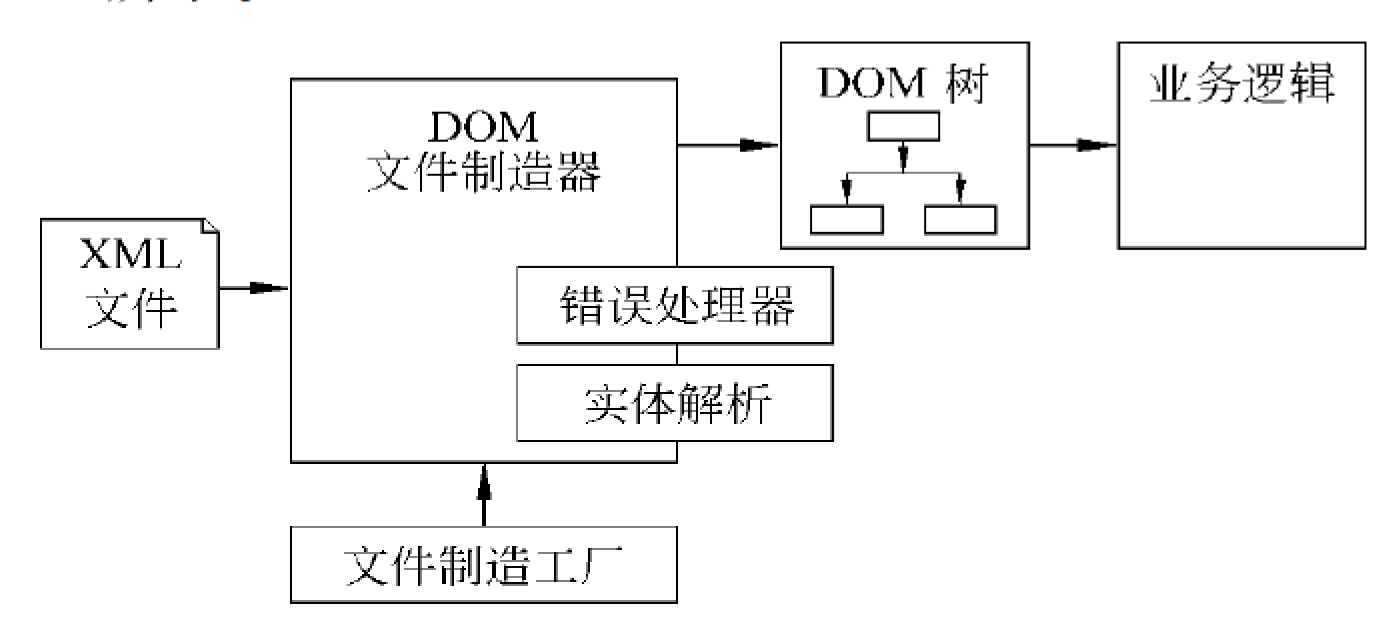


图 14-5　DOM 解析过程

1. 加载 XML 文件

加载和解析 XML 文档是创建 XML 应用的前提基础。要实现这样的功能，首先要在 JSP 中导入操作相关的包“org. w3c. dom. * ”和“javax. xml. parsers. * ”或直接导入这些包的相关类，如下所示：

```
<%@page import="org.w3c.dom.Document"%><!--导入 DOM 文件类-->
<%@page import="javax.xml.parsers.DocumentBuilderFactory"%><!--导入文件制造工厂类-->
<%@page import="javax.xml.parsers.DocumentBuilder"% ><!--导入文件制造器类-->
<%@page import="org.xml.sax.SAXException"><!--处理异常-->
```

其次，创建 DOM 对象。

```
DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();        //创建文件制造工厂对象
DocumentBuilder builder=factory.newDocumentBuilder();                       //创建文件制造器对象
Document document=builder.parse(XML 文件名);                                 //创建文件实例对象
```

2. 遍历文档结点

要遍历 DOM 对象的结点，可以根据实际需要应用 Node 接口、NodeList 接口等来实现对 DOM 对象对应的结点树的遍历。对 DOM 树遍历中必须依赖 Node 接口和 NodeList 接口，Node 接口和 NodeList 的常见方法分别如表 14-3 和表 14-4 所示。

表 14-3　org. w3c. dom. Node 接口的常见方法

方　　法	说　　明
Node appendChild(Node)	结点树末尾插入一个新结点
Node getFirstChild()	返回结点的第一个子结点
Node getLastChild()	返回结点的最后一个子结点
Node getNextSibling()	返回结点的下一个兄弟结点
String getNodeName()	返回结点的名字，取决于结点类型
String getNodeValue()	返回结点的值
short getNodeType()	返回结点的类型
Node getParentNode()	返回结点的父结点
Node getPreviousSibling()	返回结点的前一个兄弟结点
String getTextContent()	返回结点和后继的文本内容
Node removeChild(Node)	将结点从结点树中移除
Node replaceChild(Node, Node)	将参数中的前一个结点取代后一个结点
Node insertBefore(Node, Node)	将参数中的前一个结点插入，成为后一个结点的子结点

表 14-4　org. w3c. dom. NodeList 接口的常见方法

方　　法	说　　明

short getLength()	返回结点集的长度
Node item(int)	返回索引指定的结点

【例 14.1】 利用 Node 接口显示对 XML 文件所有元素的文本内容。具体内容见程序清单 14-4，运行结果如图 14-6 所示。

程序清单 14-4：

```
<!--JSP14-4.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="org.w3c.dom.*"%>
<%@page import="javax.xml.parsers.*"%>
<%@page import="org.xml.sax.*"%>
<%@page import="java.io.*"%>
<%!
   public String printNextNode(Node node){              //返回子结点的递归方法
    String result="";
      if(node.hasChildNodes())
      {
       NodeList list=node.getChildNodes();            //获取结点的所有子结点
        for(int i=0;i<list.getLength();i++){
         Node n=list.item(i);
         if(n.getNodeType()==Node.ELEMENT_NODE){       //判断是元素结点
           result+="元素->"+n.getNodeName();
           if(n.getNodeValue()==null)
            result+="<br>";
           result+=printNextNode(n);                  //返回下一个子结点
         }
        else if(n.getNodeType()==Node.TEXT_NODE)      //判断文本内容结点
           result+="    "+n.getNodeValue();
        }
           result+="<br>";
       }
       return result;
   }
%>
<html>
<head>
<title>遍历一个 XML 文件</title>
</head>
<body>
<p>
<h3>XML 遍历示例</h3>
<%
  DocumentBuilderFactory factory;
  DocumentBuilder builder;
  String fileName=
```

```
pageContext.getServletContext().getRealPath("chapter14/studentlist3.xml");
try{
    factory=DocumentBuilderFactory.newInstance();
    builder=factory.newDocumentBuilder();
    Document document=builder.parse(new File(fileName));      //创建文件实例对象
    Node root=(Node)document.getFirstChild();                 //获取根结点
    for(Node child=root;child! =null;child=child.getNextSibling()){
    //获取根结点下层所有结点
      out.println("<br>节点名"+child.getNodeName()+
                  "结点值"+child.getNodeValue()+"<br>");
      out.println(printNextNode(child));                     //输出下一个结点
      }
  }catch(SAXException e){
     out.println("发生错误!!!");
  }
%>
</p>
</body>
</html>
```

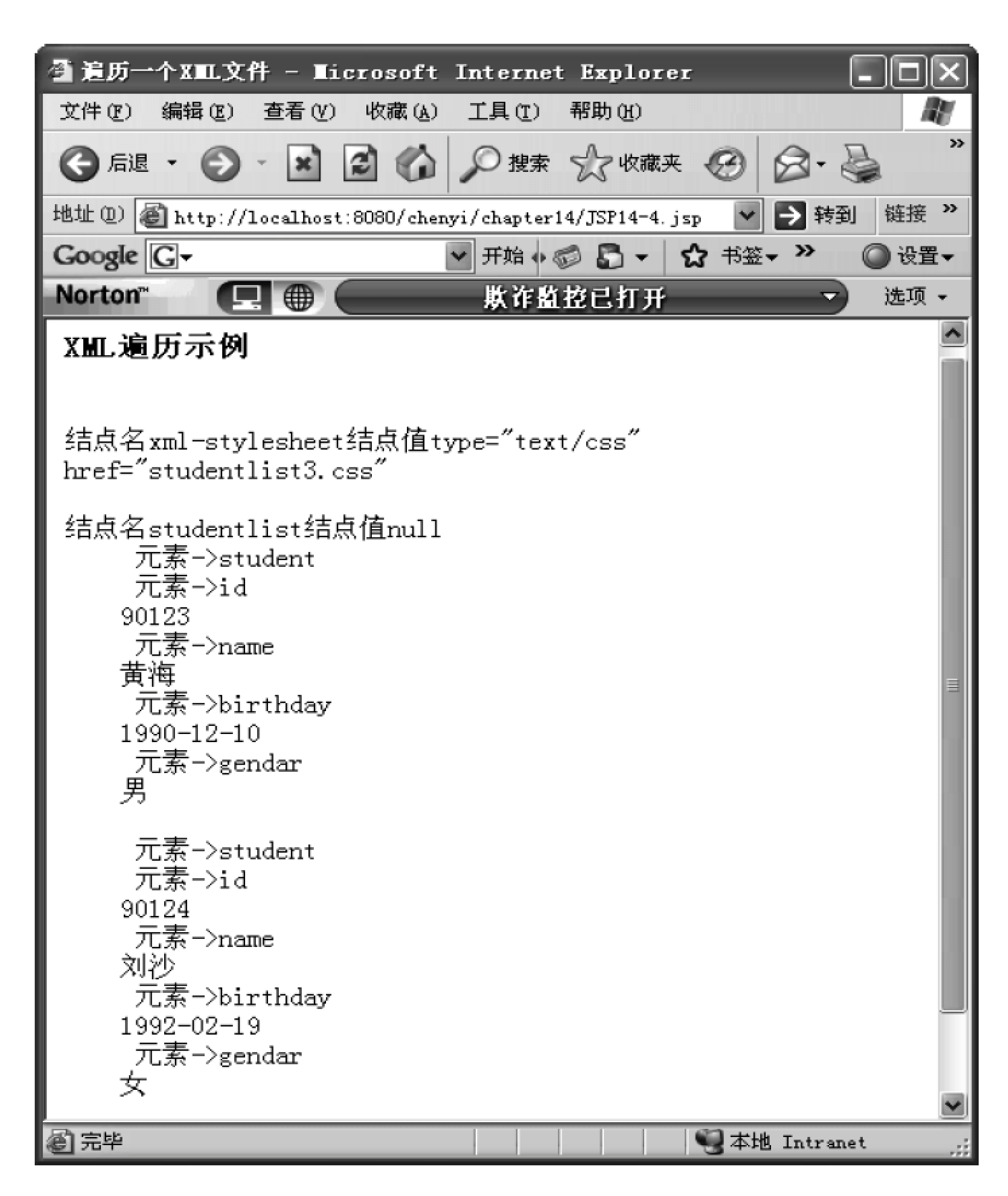


图 14-6　遍历 XML 文件的运行结果

3. 查找文档中特定的元素

通过创建 Document 接口的对象，实现对 DOM 树中特定结点的查询和访问。具体的做法是，调用 Document 对象的 getElementById() 和 getElementByTagName() 实现对 DOM 树中特定节点的访问。Document 接口的常见方法如表 14-5 所示。

表 14-5　org. w3c. dom. Document 接口的常见方法

方　　法	说　　明
Attr createAttribute(String)	创建指定名称的属性
Element createElement(String)	创建指定类型的元素
Text createTextNode(String)	创建指定字符串的文本结点
Element getDocumentElement()	直接访问文档结点的子结点
Element getElementById(String)	返回指定 ID 属性值的元素
NodeList getElementByTagName(String)	返回指定标签名的所有元素

【例 14.2】 查询一个 XML 文件的所有 name 元素包含的文本内容。具体内容见程序清单 14-5，运行结果如图 14-7 所示。

程序清单 14-5：

```
<!--JSP14-5.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="org.w3c.dom.*"%>
<%@page import="javax.xml.parsers.*"%>
<%@page import="org.xml.sax.*"%>
<%@page import="java.io.*"%>
<html>
<head>
<title>查询 XML 文件的特定元素实例</title>
</head>
<body>
<center>
<h3>XML 特定元素查询</h3>
<%
  DocumentBuilderFactory factory;
  DocumentBuilder builder;
  String fileName=
  pageContext.getServletContext().getRealPath("chapter14/studentlist3.xml");

  try{
   factory=DocumentBuilderFactory.newInstance();
   builder=factory.newDocumentBuilder();
   Document document=builder.parse(new File(fileName));    //创建文件实例对象
   NodeList list=document.getElementsByTagName("name");    //查找特定标签
   for(int i=0;i<list.getLength();i++){    //根据结点的长度依次访问结点的文本内容
```

```
            Node node=list.item(i);                    //取出第 i 个结点
            out.println(node.getNodeName()+ " "+ node.getTextContent()+"<br>");
        }
    }catch(SAXException e){
        out.println("发生错误:"+e.getMessage());
    }
%>
  </center>
</body>
</html>
```



图 14-7　查询特定元素的运行结果

4. 修改文档内容

遍历 DOM 树的目的不仅仅是将内容显示出来,最终是希望通过这样的操作过程,执行插入、删除等修改 XML 文件的结果。这些操作都涉及到对转换处理,具体处理过程如下代码所示:

```
TransformerFactory t=TransformerFactory.newInstance();   //创建转换工厂实例
Transformer transformer=t.newTransformer();              //创建转换对象
DOMSource source=new DOMSource(DOM对象);                 //转换源 DOM树
StreamResult result=new StreamResult(new File(文件名));  //创建转换结果对象
transformer.transform(source,result);                    //实现转换,产生新的文件
```

【例 14.3】 新增一个 student 元素到 studentlist3.xml 中。代码见程序清单 14-6。

程序清单 14-6:

```
<!--JSP14-6.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="org.w3c.dom.*"%>
<%@page import="javax.xml.parsers.*"%>
<%@page import="javax.xml.transform.*"%>
<%@page import="javax.xml.transform.dom.*"%>
<%@page import="javax.xml.transform.stream.*"%>
<%@page import="org.xml.sax.*"%>
```

```
<%@page import="java.io.*"%>

<html>
<head>
<title>查询 XML 文件的特定元素实例</title>
</head>
<body>
<center>
<h3>XML 特定元素查询</h3>
<%
  DocumentBuilderFactory factory;
  DocumentBuilder builder;
  String fileName="studentlist3.xml";

  try{
   factory=DocumentBuilderFactory.newInstance();
   builder=factory.newDocumentBuilder();
   Document document=builder.parse(new File(fileName));   //创建文件实例对象

   Element element=document.getDocumentElement();         //创建文档元素

   Node student=document.createElement("student");         //创建 student 元素

   Node id=document.createElement("id");                   //创建 id 元素
   Node id_Text=document.createTextNode("87687");          //创建 id 元素的文本结点
   id.appendChild(id_Text);                                //文本结点插入 id

   Node name=document.createElement("name");               //创建 name 元素
   Node name_Text=document.createTextNode("张海");
   name.appendChild(name_Text);

   Node birthday=document.createElement("birthday");       //创建 birthday 元素
   Node birthday_Text=document.createTextNode("1994-02-01");
   birthday.appendChild(birthday_Text);
   Node gendar=document.createElement("gendar");           //创建 gendar 元素
   Node gendar_Text=document.createTextNode("男");
   gendar.appendChild(gendar_Text);
   student.appendChild(id);                         //添加 student 的子结点 id
   student.appendChild(name);                       //添加 student 的子结点 name
   student.appendChild(birthday);                   //添加 student 的子结点 birthday
   student.appendChild(gendar);                     //添加 student 的子结点 gendar

   element.appendChild(student);                    //添加 student 到文挡元素中

   TransformerFactory t=TransformerFactory.newInstance();
```

```
    Transformer transformer=t.newTransformer();              //创建转换对象
    DOMSource source=new DOMSource(document);                //转换源 DOM 树
    StreamResult result=new StreamResult(new File(fileName));            //创建转换结果对象
    transformer.transform(source,result);                     //实现转换,产生新的文件
    }catch(SAXException e){
      out.println("发生错误:"+e.getMessage());
    }
%>
  </center>
</body>
</html>
```

删除与修改结点与插入类似。通过调用 Node 对象的相关方法,实现结点信息的变换。然后通过转换,将变化的 DOM 树转换写入到目的文件中。

从以上的应用例子可以发现,DOM 处理 XML 文件有着自身的特点:

(1) 将 XML 文件加载到内存中进行处理;

(2) 定义树状结构处理 XML 数据,将 XML 文件的各个成分视之为结点;

(3) 可以随时随机访问树状结构的各个结点,并可插入、删除、修改结点。DOM 方式的这些特点也带来了一些问题,就是如果 XML 文件庞大,载入到内存以及在内存中所做的处理会占用太多的内存影响执行效率。

14.2.3 JSP 应用 SAX

SAX 是 Simple API for XML 的简称。最初,SAX 是 David Megginson 用 Java 语言开发处理 XML 的 Java API。SAX 在社区讨论中不断发展完善,得到广大 Java 开发人员的支持。现在 SAX 作为标准,是实现文件解析的通用应用接口之一。

SAX 是基于事件模式,将源文件视之为事件流。具体过程是:SAX 解析器从 XML 文件中一边读取数据,一边解析已读取数据,并通知注册监听者遍历过程发生的解析事件。常见的解析事件如下:

(1) XML 文档开始;

(2) XML 文档结束;

(3) XML 文档的 DTD 事件;

(4) XML 文档的 Schema 事件;

(5) XML 元素的开始标记;

(6) XML 元素的结束标记;

(7) XML 元素的内容;

(8) 产生错误事件。

在开发人员可以根据需要编写处理这些事件的程序,来达到处理特定的事件的目的。JAXP 1.4 提供了对 SAX 2.0 的支持。JAXP 1.4 中包 org.xml.sax、org.xml.sax.ext 和 org.xml.sax.helpers 3 个包,见表 14-1。其中,org.xml.sax 是 SAX 2.0 的核心包,提供 SAX 2.0 的常见类和接口,具体内容见表 14-6。

表 14-6　org. xml. sax 包的常见类和接口

类 与 接 口	说　　明
Attributes	XML 属性表
ContentHandler	接收 XML 逻辑内容的通知的接口
DTDHandler	接收 DTD 相关的通知的接口
ErrorHandler	错误处理的基本接口
Locator	关联文档定位和 SAX 事件的接口
XMLFilter	XML 过滤器的接口
XMLReader	用回调机制处理读取 XML 文档
EntityResolver	分析实体
InputSource	XML 实体的输入源
SAXException	定义 SAX 错误与警告
SAXNotRecognizedException	SAX 不可识别的标识符的异常
SAXNotSupportedException	SAX 的不支持的操作的异常

JSP 利用 JAXP API 可以实现用 SAX 对 XML 文件的解析。要应用 SAX 解析 XML 文件，首先创建和设置一个 SAX 解析工厂，通过该工厂创建一个新解析器；然后，设置解析器的文件处理器、错误处理器、DTD 处理器和实体分析；最后通过 SAX 解析器实现对 XML 文件的解析，解析过程通知监听器发生的事件，并根据事件类型做出相应的处理，如图 14-8 所示。

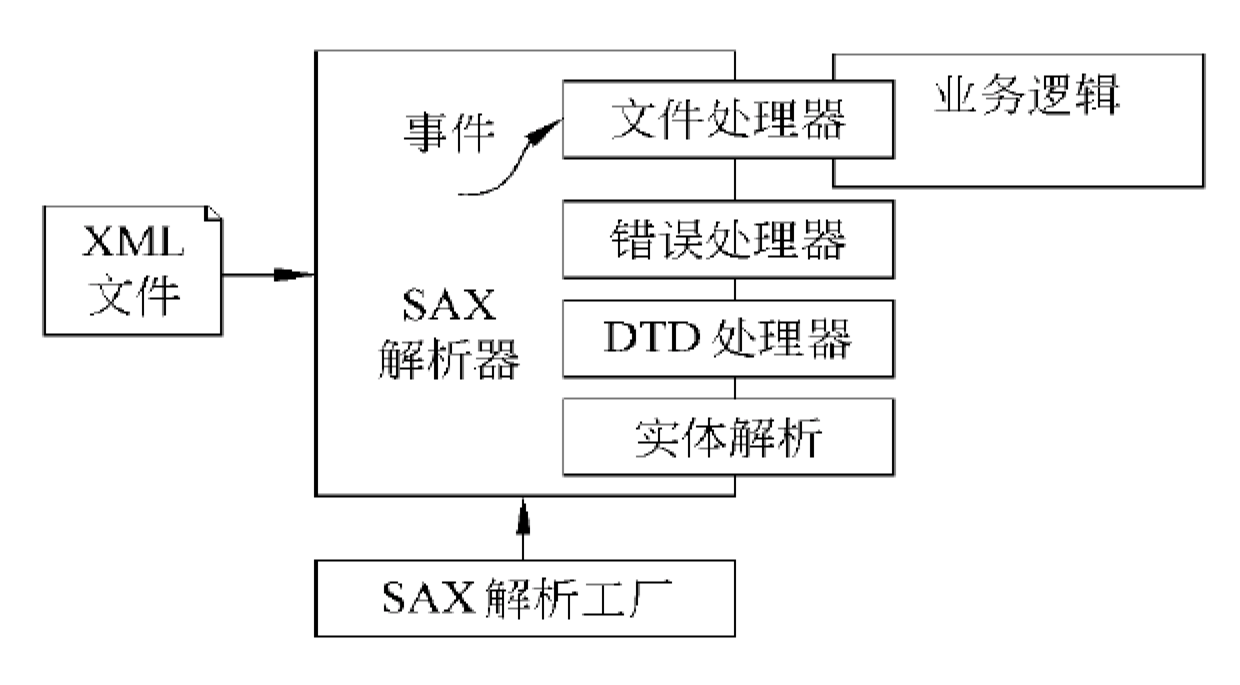


图 14-8　SAX 解析过程

【例 14.4】　利用 SAX 解析 XML 文件，将 XML 各个元素的文本信息输出。具体内容见程序清单 14-7，运行结果如图 14-9 所示。

程序清单 14-7：

```
<!--JSP14-7.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="javax.xml.parsers.*"%>
<%@page import="org.xml.sax.*"%>
<%@page import="org.xml.sax.helpers.*"%>
```

```
<%@page import="java.io.*"%>
<%!
   class SAXParserHandler extends DefaultHandler{                    //定义处理器 SAXParserHandler
      StringBuffer result=new StringBuffer();
       public void startDocument() throws SAXException{          //处理文档开始
        result.append("解析 XML 文件开始:<br>");
        result.append("<table border=1>");
        result.append("<th>结点</th><th>值</th>");
       }
       public void startElement(String namespaceURI,String localName,String qName,Attributes attrs)
       throws SAXException{                                    //处理元素的开始
          result.append("<tr><td>"+qName+"</td>");
        }
       public void characters(char[] chars,int start,int length) throws SAXException{
                                                               //处理元素的内容
          String content=new String(chars,start,length);
          result.append("<td>"+content+"</td>");              //元素中的文本信息
       }
       public void endElement(String namespaceURI,String localName,String qName)
       throws SAXException{                                    //处理元素的结束
        result.append("</tr>");
       }
       public void endDocument() throws SAXException{          //文档的结束
        result.append("</table>");
        result.append("XML 解析结束");
       }
       public StringBuffer getXMLResult(){
       return result;
       }
   }
%>
<html>
<head>
  <title>SAX 解析 XML 的应用</title>
</head>
<body>
  <center>
   <h3>SAX 解析 XML 文件</h3>
   <%   SAXParserFactory factory;
        XMLReader reader=null;
        SAXParser parser=null;
        SAXParserHandler user=new SAXParserHandler();
        String fileName=
        pageContext.getServletContext().getRealPath("chapter14/studentlist3.xml");
        try{
```

```
        factory= SAXParserFactory.newInstance();                        //创建工厂对象 factory
        parser= factory.newSAXParser();                                 //创建 SAX 解析器
        reader= parser.getXMLReader();                                  //获取 XMLReader 对象
        reader.setContentHandler(user);                                 //设置解析处理对象
        reader.parse(fileName);                                         //解析 XML 文件
      }catch(SAXException e1){
        out.println(e1.getMessage());
      }catch(IOException e2){
        out.println(e2.getMessage());
      }
      out.println(user.getXMLResult());
    %>
  </center>
  </body>
</html>
```

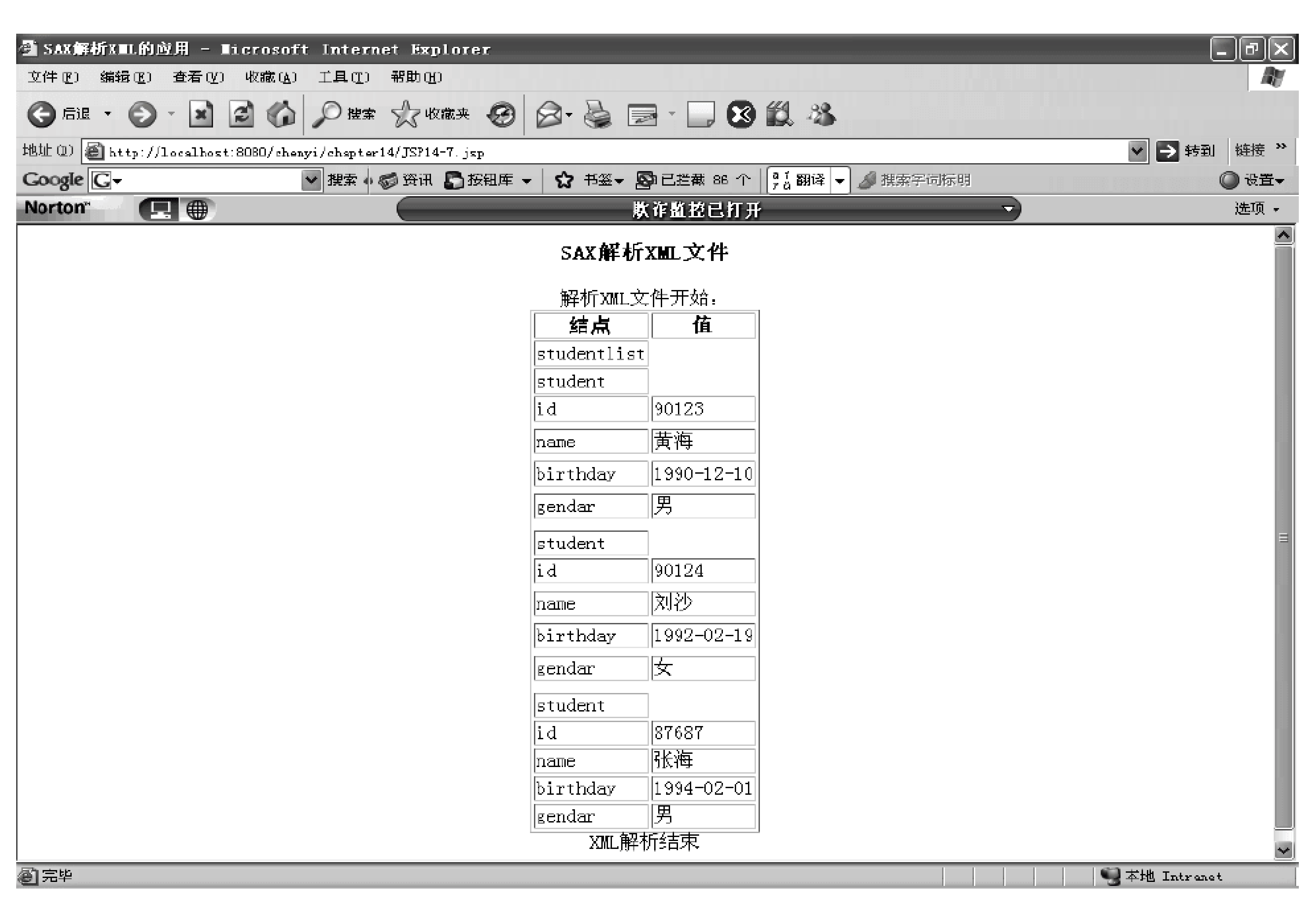


图 14-9　SAX 解析 XML 文件的运行结果

与 DOM 方式相比，SAX 2.0 通过将 XML 序列读取，同时进行解析。不需要将所有的 XML 文件一次载入内存，占据空间有限。通过回调机制，在解析过程中，通知监听器发生的事件，对发生事件的类型进行及时处理。例如，在上面的程序中，解析器利用自定义的逻辑处理组件 SAXParserHandler 的对象 user 分析 XML 文件。以 StudentList3.xml 的 id 元素为例，访问 id 元素开始标签，会调用逻辑处理对象 user 的 startElement()方法，当遍历到 id 元素的文本内容，会调用 user 的 characters()方法，当访问到 id 的结束标签，会调用 user 的 endElement()方法。这样的一个解析过程不断重复进行。可见，SAX 在解析 XML 文件上具有明显优势。但是，SAX 不能对 XML 文件直接进行编辑、修改、插入、删除，要实现这样的功能，必须结合其他方法来实现。

值得注意的是，在实际应用中，并不是在 JSP 文件中添加 Java 代码实现 XML 文件的解析。往往是开发人员编写 Java 程序，作为实用组件或 JavaBean 组件或用户自定义标签组件等达到解析 XML 文件的目的。然后由 JSP 对这些组件进行调用，实现 JSP 和 XML 的开发。

14.3 JSP 应用 XML

在前面几节中讨论，如果 JSP 中嵌套了大量的 Java 代码，无疑会影响团队之间的交流，增加开发应用的难度。在本节中将探讨 JSP 与其他技术结合，开发实现具有 MVC 结构的 XML 的应用。

通常，JSP 应用 XML 数据有两种方式：一种方式是 JSP 将 XML 数据抽取出来，封装在 JavaBean 或自定义标签中成为服务器端的对象，在 Web 应用中通过调用这些组件达到使用 XML 数据的目的；另外一种方式是 JSP 结合其他技术如 XSLT，实现 XML 数据的转换，实现特定功能。

14.3.1 JavaBean 封装 XML 数据

JavaBean 是一个可重用的组件，在具体 Web 应用的架构中通常充当保存数据的模型。实际操作中，是使用 DOM 或 SAX 解析 XML 文件，抽取开发需要的 XML 数据，将这些数据封装在 JavaBean 组件中，应用通过调用这些组件实现 Web 应用。下面的例子用 DOM 实现 XML 数据的封装，显示与程序清单 14-1 内容相近的 studentlist3. xml 的文件。

【例 14.5】 查询 studentlist3. xml 中指定编号的学生信息。

为了实现例题要求，定义一个 StudentBean. java 表示学生的 JavaBean，见程序清单 14-8。

程序清单 14-8：

```
//StudentBean.java
package beans;
import java.util.Date;
public class StudentBean {
    private String id;
    private String name;
    private Date birthday;
    private String gender;
    public Date getBirthday() {
     return birthday;
    }
    public void setBirthday(Date birthday) {
     this.birthday=birthday;
    }
    public String getGender() {
     return gender;
    }
    public void setGender(String gender) {
     this.gender=gender;
```

```
    }
    public String getId() {
      return id;
    }
    public void setId(String id) {
      this.id= id;
    }
    public String getName() {
      return name;
    }
    public void setName(String name) {
      this.name= name;
    }
}
```

本例定义 DOMStudentParser.java 程序，应用 DOM 方式解析 XML 文件。具体内容见程序清单 14-9。

程序清单 14-9：

```
//DOMStudentParser.java
package beans;
import org.w3c.dom.* ;
import javax.xml.parsers.* ;
import java.io.* ;
import org.xml.sax.* ;
import java.util.* ;
public class DOMStudentParser {                    //定义 DOM解析
     DocumentBuilderFactory factory ;
     DocumentBuilder builder;
     Document document;
  public DOMStudentParser(){
      factory= DocumentBuilderFactory.newInstance();
     try{
      builder= factory.newDocumentBuilder();
      document= builder.parse(new File("studentlist3.xml"));
     }catch(Exception e){
          System.out.println(e.getMessage());
     }
  }
  public StudentBean getRecordById(String id){           //根据记录的编号 ID查询
       StudentBean record= null;
       NodeList root= document.getElementsByTagName("student");
       for(int i= 0;i< root.getLength();i++){
            Node node= root.item(i);
            for(node= node.getFirstChild();
            node.getNodeType()! =Node.ELEMENT_NODE;
            node= node.getNextSibling());                 //查询结点元素
```

```
            if(node.getTextContent().equals(id)){          //根据编号查询
                record=new StudentBean();                  //将查询结果保存到record中
                String[] r=new String[4];
                int j=0;
                for(Node tmp=node;tmp! =null&&j<4;tmp=tmp.getNextSibling()){
                    if(tmp.getNodeType()==Node.ELEMENT_NODE){
                    r[j]=new String(tmp.getTextContent().trim());
                    System.out.println(r[j++]);
                    }
                }
                record.setId(r[0]);
                record.setName(r[1]);
                try{   java.text.SimpleDateFormat   df   =
                   new   java.text.SimpleDateFormat("yyyy-MM-dd");
                java.util.Date   date   =   df.parse(r[2]);              //将字符串转换成日期
                record.setBirthday(date);
                }catch(Exception e3){
                }
                record.setGender(r[3]);
                return record;
            }
        }
        return record;
    }
}
```

定义一个 JSP 页面，显示查询的结果。具体内容见程序清单 14-10，运行结果如图 14-10 和图 14-11 所示。

程序清单 14-10：

```
<!--JSP14-10.jsp-->
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@page import="beans.*"%>
<html>
<head>
<title>查询 XML 文件的特定元素实例</title>
</head>

<body>
<center>
<h3>按编号查询学生记录</h3>
<form action="" method="post">
  编号：<input type="text" name="stud_id"/>
  <input type="submit" value="查询"/>
</form>
<%
   String id=request.getParameter("stud_id");
```

```
    if(id==null) id="";
    if(! id.equals("")){
      DOMStudentParser  parser=new DOMStudentParser();       //创建解析对象
      StudentBean bean=parser.getRecordById(id);              //查询特定编号的记录
%>
     <h4>查询结果</h4><br/>
<%
      if(bean==null)                                          //查找记录不存在
        out.println("编号为:"+id+"的记录不存在!");
      else{                                                   //查找记录存在,利用表格输出
  %>
      <table border="1">
        <tr><td>编号</td>
            <td><%=bean.getId()%></td>
       </tr>
       <tr><td>姓名</td>
           <td><%=bean.getName()%></td>
       </tr>
       <tr><td>出生日期</td>
           <td><%=bean.getBirthday()%></td>
       </tr>
       <tr><td>性别</td>
           <td><%=bean.getGender()%></td>
       </tr>
  <%     }
      }
%>
</center>
</body>
</html>
```



图 14-10 查询记录不存在



图 14-11　查询结果存在

14.3.2　用户自定义标签封装 XML

实现 JSP 页面与 Java 代码的分离是开发 JSP 的一个重要内容。在第 12 章，介绍了用 JavaBean 技术封装了大量的 Java 代码实现业务逻辑或封装数据模式。但是，在应用 JavaBean，有时还需要通过编写 Java 代码来实现 JSP 页面，并没有将 Java 代码和 JSP 页面完全地分开。JSP 的用户自定义标签可以提供实现 JSP 页面与 Java 代码分离的又一处理方法。

用户自定义标签是一种特殊形式的 JavaBean 组件。通过用户自定义标签允许用户自定义基于 XML 的标签。有效解决了美工与技术小组之间的交流理解问题。程序员开发实现特定功能的标签，美工只需要理解和将这些标签嵌入到 JSP 页面就可以，有效地提高了工作效率。同时，增加了页面的可读性。

要开发用户自定义标签，有 3 个步骤：

(1) 编写处理标签 Java 类；

(2) 创建描述标签库的文件；

(3) 在 JSP 文件导入标签库，以及应用标签库的标签。

下面将通过分别查询“studentlist3.xml”中的所有性别为“男”和“女”的记录的应用实例来了解用户自定义标签封装 XML 数据的过程。

(1) 编写处理标签的类。这些类的作用是实现对于用户自定义标签的处理方案。这些 Java 类必须扩展 javax.servlet.jsp.TagSupport 或 javax.servlet.jsp.BodyTagSupport 的类。其中，BodyTagSupport 是 TagSupport 的子类。通常利用 TagSupport 类实现它们的常见方法见表 14-7 和表 14-8。

表 14-7　javax. servlet. jsp. TagSupport

方　　法	说　　明
int doAfterBody()	默认处理标签主体
int doStartTag()	默认处理自定义标签的开始标志，返回 SKIP_BODY
int doEndTag()	默认处理自定义标签的结尾标志，返回 EVAL_PAGE
Tag getParent()	返回封闭标签的上级标签
void setParent(Tag)	设置封闭标签的上级标签
void setPageContext(PageContext)	设置 PageContext 对象，在 doStartTag()调用之前调用
Object getValue(String)	获取参数对应的值
void setValue(String,Object)	设置名字-值对

表 14-8　javax. servlet. jsp. BodyTagSupport

方　　法	说　　明
int doAfterBody()	标签主体后执行
int doStartTag()	默认处理自定义标签的开始标志，返回 SKIP_BODY
int doEndTag()	默认处理自定义标签的结尾标志，返回 EVAL_PAGE
void doInitBody()	准备评估标签主体
BodyContent getBodyContent()	获取当前标签主体内容
void setBodyContent()	准备评估标签的实体
void release()	释放状态

下列定义了一个处理自定义标签的类 SAXParserTag，当 JSP 容器遇到标签的开始标志，就会调用 doStartTag()方法。具体内容见程序清单 14-11。

程序清单 14-11：

```
//SAXParserTag.java
package tags;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
public class SAXParserTag extends TagSupport {              //处理自定义标签
    private String gender;                                  //标签属性 gender;
public int doStartTag()throws JspException{
    try{
     SAXParserBuilder builder=new SAXParserBuilder();
     if(gender!=null&&gender.equals("女"))
       builder.setCondition(gender);
     pageContext.getOut().println(builder.getRecords());
    }catch(IOException e){
```

```
            throw new JspException(e.getMessage());
        }
         return SKIP_BODY;                              //忽略标签体
    }
    public String getGender() {                         //获取性别属性
        return gender;
    }
    public void setGender(String gender) {              //设置性别属性
        this.gender=gender;
    }
}
```

类 SAXParserTag 是利用自定义 SAXParserBuilder 类来解析 XML 文件。SAXParserBuilder 中定义了一个内部类 SAXHandler 来解析 XML 文件，并利用 SAX2.0 API 执行按性别查询记录。具体内容见程序清单 14-12。

程序清单 14-12：

```
//SAXParserBuildre.java
package tags;
import javax.servlet.jsp.* ;
import javax.xml.parsers.* ;
import org.xml.sax.* ;
import org.xml.sax.helpers.DefaultHandler;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class SAXParserBuilder {
        private StringBuffer result;
        private SAXParserFactory factory;
        private SAXHandler handler;
        private String fileName;
        private String condition;
        public SAXParserBuilder(){
            result=new StringBuffer("");
            fileName=" studentlist3.xml";
            condition=new String("男 ");        //设置标签默认查询条件为性别="男"
        }
        public void parseToRecords()throws JspException{
            try{
                factory=SAXParserFactory.newInstance();
                SAXParser parser=factory.newSAXParser();
                XMLReader reader=parser.getXMLReader();
                handler=new SAXHandler();
                reader.setContentHandler(handler);
                reader.parse(fileName);
```

```
        }catch(Exception e){
            System.out.println(e.getMessage());
            throw new Error("警告:严重错误!"+e.getLocalizedMessage());
        }
    }
    public void setCondition(String condition){                //设置查询条件
        this.condition=condition;
    }
    public String getCondition(){                              //获取查询条件
        return this.condition;
    }
    public StringBuffer getRecords()throws JspException{
        parseToRecords();                                      //解析 XML 文件
        result.append(handler.getRecords());                   //添加查询结果
        return result;
    }
    class SAXHandler extends DefaultHandler{                                  //内部类,SAX 解析处理
        StringBuffer record;
        StringBuffer s;
        String id;
        String name;
        Date birthday;
        String gender;
        public void startDocument()throws SAXException{
        record=new StringBuffer();
        record.append("<table border='1'><tr>");
        record.append("<th>编号</th><th>姓名</th>"+
                                    "<th>出生日期</th><th>性别</th>");
            record.append("</tr>");
        }
      public void startElement(String namespaceURI,String localName,String qName,Attributes attrs)
throws SAXException{                                           //处理元素的开始
            s=new StringBuffer();
        }
        public void characters(char[] chars,int start,int length)throws SAXException{
        //处理元素的内容
          s.append(chars,start,length);
        }
      public void endElement(String namespaceURI,String localName,String qName)
          throws SAXException{                                 //处理元素的结束
            String text=s.toString().trim();
          if(qName.equals("id"))
          id=text;
          else if(qName.equals("name"))
          name=text;
```

```
                    else if(qName.equals("birthday")){
                    SimpleDateFormat df=
                                new   java.text.SimpleDateFormat("yyyy-MM-dd");
                    try{
                    birthday=df.parse(text);
                    }catch(ParseException e2){
                      throw new SAXException("日期解析出现错误"+e2.getMessage());
                    }
                    }
                    else if(qName.equals("gender")){
                    gender=text;
                    if(gender.equals(getCondition())){
                        record.append("<tr><td>"+id+"</td>"+
                                            "<td>"+name+"</td>"+
                                            "<td>"+birthday+"</td>"+
                                            "<td>"+gender+"</td></tr>");
                    }
                    }
                  }
                  public void endDocument() throws SAXException{
                  //文档的结束
                   record.append("</table>");
                  }
                  public StringBuffer getRecords(){
                  return record;
                  }
                }//end 内部 class SAXHandler
        }//结束 SAXParserBuilder 类
```

（2）创建标签库描述文件。标签库描述文件实质上是一个 XML 文件，只是它的文件扩展名为“.tld”。描述一个标签库要通过 taglib 元素来实现。由于标签库下可以有一个或多个标签，这些描述标签库文件中用 tag 元素表示这些标签。标签可以根据需要定义属性，可以用 attribute 元素来描述标签的属性。程序清单 14-13 展示了一个标签库描述文件 MyTagLib.tld。

程序清单 14-13：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--MyTagLib.tld-->
<taglib>                                   <!--定义标签库元素 -->
  <tlib-version>1.0</tlib-version>         <!--指定标签库的版本 -->
  <jsp-version>1.2</jsp-version>           <!--指定 JSP 的版本 -->
  <shortname>mytablib</shortname>          <!--指定标签库默认的前缀 -->
  <uri>/WEB-INF/MyTagLib</uri>             <!--设置标签库的惟一访问标识符 -->
  <info>我的自定义标签库</info>            <!--定义标签库的说明信息 -->
  <tag>                                    <!--定义标签元素 -->
```

```
        <name>parser</name>                    <!--定义标签名 parser-->
        <tag-class>                            <!--指定标签的处理类 -->
          tags.SAXParserTag
        </tag-class>
        <body-content>                         <!--定义标签的主体内容 -->
          empty
        </body-content>
        <info>查询学生记录</info>              <!--定义标签的说明信息-->
        <attribute>                            <!--定义标签元素的属性-->
            <name>gender</name>                <!--定义属性名 gender-->
            <required>false</required>         <!--指定属性是否必须-->
            <rtexprvalue>                      <!--指定属性值是否是 request-time 表达式-->
                true
            </rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

(3) JSP 页面应用标签库。JSP 文件中要应用标签库,有两个步骤:

① 用指令<%@taglib>声明对标签库的引用。具体的用法如下:

```
<%@ taglib uri="标签库的唯一访问标识符" prefix="引用标签库的前缀名"%>
```

其中,通过 uri 指定的标签库的唯一访问标识符必须与标签库中的 uri 元素内容一致。

② 在 JSP 页面中用前缀名来访问标签,形式如下:

```
<前缀名:标签名 [属性名="属性值"…] />
```

注意:“[ ]”部分表示可选。

下列程序清单 14-14 定义了 JSP14-13.jsp,来应用自定义标签实现查询结果输出,运行结果如图 14-12 所示。

程序清单 14-14:

```
<!--JSP14-14.jsp-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ page contentType="text/html; charset=gb2312" language="java"%>
<%@ taglib uri="/WEB-INF/MyTagLib" prefix="mytag"%>
<html>
  <head>
    <title>查询 XML 数据</title>
  </head>
   <body>
   <center>
    <h3>查询性别为男的所有学生</h3><br/>
    <mytag:parser gender="男" /><br/>
    <hr/>
    <h3>查询性别为女的所有学生</h3><br/>
```

```
      <mytag:parser gender="女"/>
    </center>
  </body>
</html>
```



图 14-12　自定义标签的运行结果

（4）web.xml 部署标签库。要让 JSP 自定义标签发挥作用，必须要部署标签库。即，在 WEB-INF 目录下的 web.xml 中定义 taglib 元素，对自定义标签进行说明和设置。

程序清单 14-15：

```
<?xml version="1.0" encoding="UTF-8"?>
<!--web.xml-->
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <jsp-config>
    <taglib>
     <taglib-uri>/WEB-INF/MyTagLib</taglib-uri>
        <!--引用标签库的唯一访问标识符-->
      <taglib-location>/WEB-INF/MyTagLib.tld</taglib-location>
        <!--引用标签库的标签的所在位置-->
    </taglib>
  </jsp-config>
</web-app>
```

从上述的例子中可以看出，自定义标签库的应用可以降低 JSP 页面的复杂程度，使得

JSP 页面的维护更为容易，并使得代码重用得以提高。在 JSP 中还支持 JSP 标准标签库(JSTL)，将代码重用最大化，由于篇幅限制，本节并不介绍，请查看相关资料。

14.3.3 JSP 转换 XML 文件

XSLT 可以直接将 XML 文件转换成特定格式。但是，并不是所有的浏览器都支持 XML 文件。例如 Internet Explorer 5.0 以下版本的浏览器就并不支持 XSLT 直接转换 XML 文件。为了提高可用性，需要在服务器端先用 XSLT 转换 XML，再将转换好形式的文件发给用户。

JSP 可以实现在服务器端用 XSLT 直接转换 XML 文件。要实现这样的转换，首先，要为 JSP 的 Web 应用下载 taglibs-xsl.jar、xalan.jar 和 xerces.jar，将这些压缩文件放置在 WEB-INF\lib 目录下。然后，采用 JSP 调用外部 JSP＋XML 标签库方式来实现，这需要在 JSP 文件用“＜％＠taglib uri＝"http://jakarta.apache.org/taglibs/xsl-1.0" prefix＝"前缀名"％＞”指令设置标签库。

JSP＋XML 标签库定义了 4 种实现转换的方式：：“标签库 apply 标签导入”、“直接嵌入 XML 数据”、“利用标签库的 include 标签导入”和“用标签库的 import 标签导入”。

1．利用标签库 apply 动作标签

这是 jakarta 项目的推荐方式。利用 jakarta 项目组定义好的 JSP＋XML 标签库，用标签 apply 指定 XML 文件和 XSLT 文件，形如“＜前缀名:apply xml＝"xml 文件" xsl＝"xsl 文件"/＞”来实现 XSLT 文件对 XML 转换。

【例 14.6】 利用标签库实现转换的应用实例。具体内容见程序清单 14-16。

程序清单 14-16：

```
<!--JSP14-16.jsp-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@taglib uri="http://jakarta.apache.org/taglibs/xsl-1.0" prefix="xsltr"%>
<html>
  <head>
    <title>JSP 页面用 XSLT 转换 XML 文件</title>
  </head>
  <body>
   <center>
    <h3>XSLT 转换 XML 文件</h3><br/>
    <xsltr:apply xml="/XML5-16.xml" xsl="/XSLT5-16.xsl"/><br/>
    <hr/>
   </center>
  </body>
</html>
```

2．直接嵌入 XML 数据

直接嵌入 XML 数据就是将 XML 数据直接嵌套在 JSP 页面中，通过调用外部标签库的标签 apply 指定的 XSLT 文件实现转换。这种方式是由 JSP 页面本身处理转换过程。

【例 14.7】 JSP 页面直接嵌入 XML 数据实现转换的应用实例。具体内容见程序清单 14-17。

程序清单 14-17：

```
<!--JSP14-17.jsp-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@page contentType="text/html; charset=gb2312" language="java"%>
<%@taglib uri="http://jakarta.apache.org/taglibs/xsl-1.0" prefix="xsltr"%>
<html>
  <head>
    <title>JSP 页面用 XSLT 转换 XML 文件</title>
  </head>
   <body>
   <center>
    <h3>XSLT 转换 XML 文件</h3><br/>
    <xsltr:apply xsl="/5-16.xsl">
      <?xml version="1.0" encoding="UTF-8"?>
        <mailbox>
          <email id="liu1">
            <from>liu@yahoo.com.cn</from>
            <to catalog="friend">wang@yahoo.com.cn</to>
            <subject>hello</subject>
            <message>Hello,contact with me tomorrow</message>
          </email>
          <email id="liu2">
          <from>wang@yahoo.com.cn</from>
          <to catalog="family">wang_he@yahoo.com.cn</to>
          <subject>hello</subject>
          <message>8:00 by train.</message>
        </email>
        <email id="liu3">
          <from>wang@yahoo.com.cn</from>
          <to catalog="job">he_jiang@yahoo.com.cn</to>
          <subject>apply for engineer</subject>
          <message>wait for reply</message>
        </email>
      </mailbox>
    </xsltr:apply>
    <hr/>
   </center>
  </body>
</html>
```

3. 利用标签库的 include 标签导入

它是用外部的 JSP+XML 标签库的 include 标签实现对 XML 数据的导入。形式如下：

```
< 前缀名 :apply xsl= "XSLT 文件 ">
    < 前缀名 :include page= "XML 文件 "/>
< /前缀名 :apply>
```

与上面的两种方式不同在于,这种方式是通过 JSP 容器启动另外一个 Servlet 或 JSP 页面来实现转换。这种方式可以实现导入的同一个 XSLT 文件对不同的 XML 文件的转换。

【例 14.8】 JSP 页面用 include 标签实现转换的应用实例。具体内容见程序清单 14-18。

程序清单 14-18:

```
<!--JSP14-18.jsp-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://jakarta.apache.org/taglibs/xsl-1.0" prefix="xsltr"%>
<html>
  <head>
    <title>JSP 页面用 XSLT 转换 XML 文件</title>
  </head>
   <body>
   <center>
    <h3>XSLT 转换 XML 文件</h3>
     <xsltr:apply xsl="/XSLT5-16.xsl">
        <xsltr:include page="/XML5-16.xml"/>
     </xsltr:apply>
    <hr/>
   </center>
  </body>
</html>
```

4. 用标签库的 import 标签导入

用标签库的 import 标签导入 XML 文件,然后用 XSLT 去转换。使用 import 标签的形式如下:

```
<前缀名 :import id="标号名 " page="xml 文件 "/>
   <前缀名 :apply nameXml="标号名 " xsl="xsl 文件 "/>
```

导入的 XML 文件的一个重要特点,就是作用范围在 Page,表示请求的当前页面。

【例 14.9】 JSP 页面用 import 标签实现转换的应用实例。具体内容见程序清单 14-19。

程序清单 14-19:

```
<!--JSP14-19.jsp-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://jakarta.apache.org/taglibs/xsl-1.0" prefix="xsl"%>
<html>
  <head>
    <title>JSP 页面用 XSLT 转换 XML 文件</title>
```

```
    </head>
     <body>
     <center>
      <h3>XSLT转换 XML 文件</h3>
       <xsl:import id="data" page="/XML5-16.xml"/>
          <xsl:apply nameXml="data" xsl="/XSLT5-16.xsl"/>
      <hr/>
     </center>
    </body>
</html>
```

比较 4 种运行结果,可以发现前两种方式的中文内容不会乱码(如图 14-13 所示),而后两种方式则出现乱码(如图 14-14 所示)。这是因为,前两种方式是在 JSP 本身页面实现转换,而后两种方式是由 JSP 容器生成不同的 Servlet 处理 JSP 页面和 XSLT 转换 XML。



编号	发信地址	收信地址	类别	主题	内容
liu1	liu@yahoo.com.cn	wang@yahoo.com.cn	friend	hello	Hello,contact with me tomorrow
liu2	wang@yahoo.com.cn	wang_he@yahoo.com.cn	family	hello	8:00 by train.
liu3	wang@yahoo.com.cn	he_jiang@yahoo.com.cn	job	apply for engineer	wait for reply

图 14-13　前两种方式运行结果



??	????	????	??	??	??
liu1	liu@yahoo.com.cn	wang@yahoo.com.cn	friend	hello	Hello,contact with me tomorrow
liu2	wang@yahoo.com.cn	wang_he@yahoo.com.cn	family	hello	8:00 by train.
liu3	wang@yahoo.com.cn	he_jiang@yahoo.com.cn	job	apply for engineer	wait for reply

图 14-14　后两种方式的运行结果

14.4　JSP+XML 的应用实例：开发技术论坛

XML 数据关系明了、简单。更重要的是 XML 在不同平台下都可以得到支持。对于一些简单 Web 应用是可以完全利用 XML 作为后台的数据源进行开发，而不依赖数据库。本节将介绍 JSP 与 XML 结合开发一个具有基本论坛功能的 JSP+XML　BBS 技术论坛。

14.4.1　技术论坛简介

JSP+XML BBS 论坛是实现计算机技术问题的讨论的公告板，用户分成两类：客户和管理员。客户需要通过注册，然后登录才具有访问论坛的权限。客户可以分类浏览信息、发表信息、对一些问题进行回复。管理员也可以实现对论坛删除、论坛的创建等管理功能。JSP+XML BBS 论坛的采用了 3 层软件结构：客户层、Web 服务器层和数据源。如图 14-15 所示。

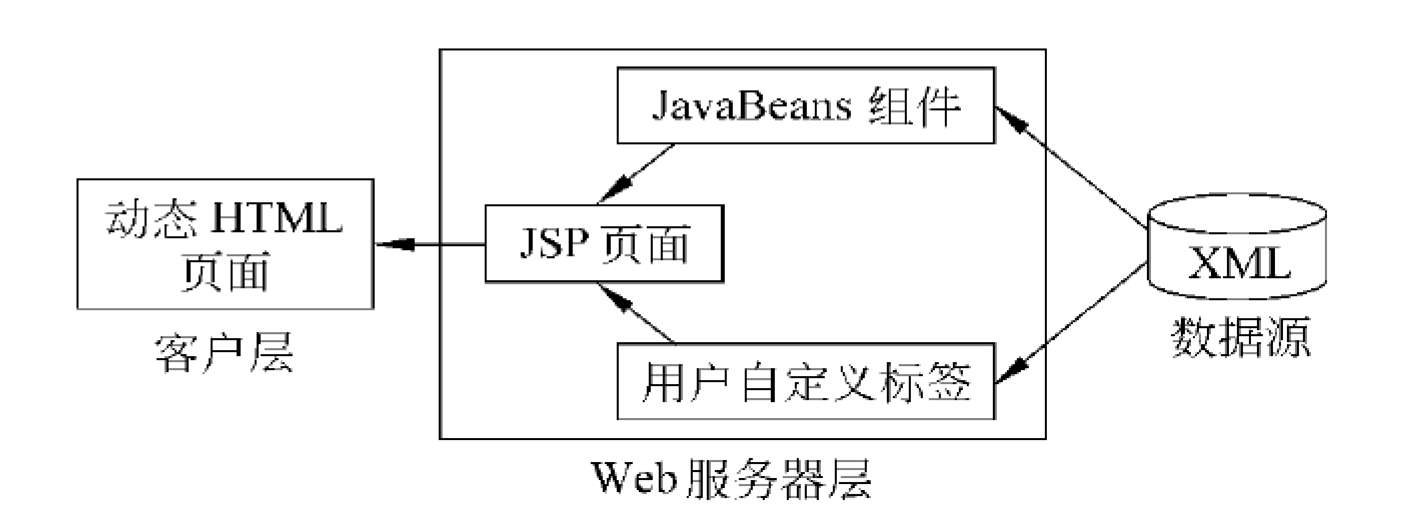


图 14-15　JSP+XML BBS 的软件结构

(1) 客户层。是用户通过浏览器动态访问 HTML 页面，实现与 Web 服务器层的交互。

(2) Web 服务器层。包含用户自定义标签、Servlet 和 JSP 等组件。这些组件运行在 Web 层，各行其责。这些组件处理用户的请求，执行业务逻辑处理，并从数据源中获取必要的数据，动态页面的生成，响应用户的请求。

(3) 数据源。保存用于处理 Web 应用的数据。

14.4.2　用户登录

JSP+XML BBS 论坛中，用户和管理员登录只需要输入已注册的 E-mail 和密码。只要查询到记录，用户就直接进入到论坛导航中。如果是管理员就直接进入到管理论坛界面。如果输入信息不正确，则客户会直接导航到注册界面注册。

1. 保存用户信息的 XML 文件

用 userlist.xml 可以表示多个客户和管理员的信息。如程序清单 14-20 所示。

程序清单 14-20：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--userlist.xml-->
<users>
<manager>    <!--表示管理员-->
    <email>chan@yahoo.com</email>
```

```
    <password>123456</password>
    <name>管理员</name>
  </manager>
  <user>      <!--表示用户-->
    <email>yi@yahoo.com</email>
    <password>123456</password>
    <name>Mountain</name>
    <level>高</level>
    ⋮
  </user>
  <user>       <!--表示用户-->
    <email>jiang@yahoo.com</email>
    <password>123456</password>
    <name>See food</name>
    <level>用户</level>
    ⋮
</user>
</users>
```

2. 登录界面

登录界面如图 14-16 所示。在实际运行中,管理员与用户的登录入口不同。本节为了简洁,将两者处理放置在一个页面 index.jsp 中,具体内容见程序清单 14-21。

程序清单 14-21:

```
<!--index.jsp-->
<%@page contentType="text/html;charset=gb2312"%>
<%@page language="java" import="java.io.*"%>
<html>
<head>
  <title>用户登录</title>
  <link href="resource/style.css" rel="stylesheet" type="text/css">
</head>

<body>
  <h3>JSP+XML BBS论坛系统</h3>
    <form method="get" action="LoginServlet">
      <table>
        <tr>
          <td>登录 Email</td>
          <td><input type="text" name="email"/></td>
        </tr>
        <tr>
          <td>密码</td>
          <td><input type="password" name="password"/></td>
        </tr>
        <tr>
```

```
            <td/>
            <td><input type="radio" name="identity" value="manager"/>管理员
                <input type="radio" name="identity" value="user"/>用户
            </td>
        </tr>
        <tr>
            <td/>
            <td><input type="submit" name="action" value="Login"/>
                <input type="submit" name="action" value="Register"/>
            </td>
      </table>
    </form>
</body>
</html>
```



图 14-16　登录界面运行结果

3. 登录信息验证

定义 LoginServlet.java，用 DOM API 实现对 userlist.xml 文件信息查询。用户登录成功导航至 user.jsp，管理员登录成功导航至 manager.jsp。否则导航到 register.jsp 进行注册。具体内容见程序清单 14-22。

程序清单 14-22：

```
//LoginServlet.java
package servlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
public class LoginServlet extends HttpServlet{
    String username;
```

```
public void doGet(HttpServletRequest request,HttpServletResponse response)
  throws ServletException,IOException{
    doPost(request,response);
}
public void doPost(HttpServletRequest request,HttpServletResponse response)
   throws ServletException,IOException{
    String email=request.getParameter("email");              //获取 E-mail
    String password=request.getParameter("password");        //获取 password
    String identity=request.getParameter("identity");        //获取客户的身份
    String action=request.getParameter("action");            //获取用户动作
    if(email==null) email="";
    if(password==null) password="";
    if(identity==null) identity="";
    if(action==null) action="";
    RequestDispatcher  view;
    if(action.equals("Login")&&checkIdentity(email,password,identity)){
     view=request.getRequestDispatcher(identity+".jsp");              //根据身份进行不同处理
     if(username!=null){
             HttpSession session=request.getSession();
             session.setAttribute("username", username);              //将用户昵称写入会话
             session.setAttribute("email", email);                    //将用户 E-mail 写入会话
         }
   view.forward(request, response);
    }
      else{
        view=request.getRequestDispatcher("register.jsp");
      view.forward(request, response);
      }
}
public boolean checkIdentity(String email,String password,String identity){
                                                             //验证客户信息
      DocumentBuilderFactory factory;
      DocumentBuilder builder;
      String fileName="userlist.xml";
     try{
       factory=DocumentBuilderFactory.newInstance();
       builder=factory.newDocumentBuilder();
       Document document=builder.parse(new File(fileName));                 //创建文件实例
       NodeList list;
       if(identity.equals("manager"))
         list=document.getElementsByTagName("manager");
       else
         list=document.getElementsByTagName("user");
       for(int i=0;i<list.getLength();i++){             //遍历
            Node node=list.item(i).getFirstChild();
```

```
                    while(node!=null){
                        if(node.getNodeName().equals("email")){
                          if(! (node.getTextContent().equals(email)))
                                break;
                          }
                        else if(node.getNodeName().equals("password")){
                              if(!(node.getTextContent().equals(password)))
                                  break;
                          }
                        else if(node.getNodeName().equals("name")){
                            username=node.getTextContent();
                            return true;
                        }
                             node=node.getNextSibling();
                    }
                }
            }catch(Exception e){
                System.out.println(e.getMessage());
                if(e!=null)   e.printStackTrace();
                }
            return false;
        }
    }
```

14.4.3 用户注册

register.jsp 提供用户注册的界面，提供了用户填写注册信息，具体内容见程序清单 14-23，运行结果如图 14-17 所示。然后，将用户提交的注册信息到 RegisterServlet 中将注册的信息插入到 userlist.xml 中。注册信息成功后，页面会自动转向到主题论坛中。RegisterServlet.java 见程序清单 14-24。

程序清单 14-23：

```
<!--register.jsp-->
<%@page contentType="text/html; charset=UTF-8" language="java"%>
<html>
<head>
   <title>JSP+XML 论坛</title>
   <link href="resource/style.css" rel="stylesheet" type="text/css">
</head>
<body>
   <center>
   <h3><a href="user.jsp">JSP+XML BBS 论坛系统</a></h3>
   <form action="RegisterServlet" method="get">
     <table>
      <caption>用户注册</caption>
```

```
        <tr>
            <td>E-mail<input type="text" name="email"/> * </td>
        </tr>
        <tr>
            <td>用户名<input type="text" name="clientName"/> *
       </td>
        </tr>
        <tr>
            <td>密码<input type="password" name="password"/> *
       </td>
        </tr>
        <tr>
            <td>
                <input type="submit" value="注册"/>
                <input type="reset" value="重置"/>
            </td>
        </tr>
       </table>
    </form>
    </center>
</body>
</html>
```



图 14-17 用户注册界面

程序清单 14-24：

```
//RegisterServlet.java
package servlet;
import java.io.* ;
import javax.servlet.* ;
import javax.servlet.http.* ;
```

```
import org.w3c.dom.* ;
import javax.xml.parsers.* ;
import javax.xml.transform.* ;
import javax.xml.transform.dom.* ;
import javax.xml.transform.stream.* ;
public class RegisterServlet extends HttpServlet {
    private String email;
    private String password;
    private String name;
    public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException{
        doPost(request,response);
    }
    public void doPost(HttpServletRequest request,HttpServletResponse response)
    throws ServletException,IOException{
        if(initInfo(request)){                              //接收注册信息成功与否
         insertAElement();                                  //插入用户注册信息
        HttpSession session= request.getSession();
        session.setAttribute("email", email);
        session.setAttribute("username",name);
        }
        RequestDispatcher view= request.getRequestDispatcher("user.jsp");
    view.forward(request, response);
    }
    public boolean initInfo(HttpServletRequest request){     //接收用户注册信息
        email= request.getParameter("email");
        password= request.getParameter("password");
        name= request.getParameter("clientName");
        if(email==null||password==null||name==null) return false;
        return true;
    }
    public Node getATopic(Document document){                //建立新结点 user
        Element userNode= document.createElement("user");

        Node emailNode= document.createElement("email");
        emailNode.appendChild(document.createTextNode(email));
        Node passwordNode= document.createElement("password");
        passwordNode.appendChild(document.createTextNode(password));
        Node nameNode= document.createElement("name");
        nameNode.appendChild(document.createTextNode(name));
        Node levelNode= document.createElement("level");
        levelNode.appendChild(document.createTextNode("用户"));
        userNode.appendChild(emailNode);
        userNode.appendChild(passwordNode);
        userNode.appendChild(nameNode);
```

```
            userNode.appendChild(levelNode);
            return userNode;
        }
        public void insertAElement(){                          //插入新结点到 userlist.xml 中。
            DocumentBuilderFactory factory;
            DocumentBuilder builder;
            String fileName=" userlist.xml";
            try{   factory=DocumentBuilderFactory.newInstance();
                   builder=factory.newDocumentBuilder();
                   Document document=builder.parse(new File(fileName));
                   Element element=document.getDocumentElement();
                   element.appendChild(getATopic(document));
                   TransformerFactory t=TransformerFactory.newInstance();
                   Transformer transformer=t.newTransformer();
                   DOMSource source=new DOMSource(document);
                   StreamResult result=new StreamResult(new File(fileName));
                   transformer.transform(source,result);
            }catch(Exception e){
                 System.out.println(e.getMessage());
            }
        }
    }
```

14.4.4 论坛导航

对于用户登录成功后,进入显示多个主题论坛的导航页面。用户可以在系统中选择论坛。论坛的主要属性有论坛名(name)、版主(manager)、论坛描述(description)。多个论坛基本信息保存在一个论坛列表 forumlist.xml 中,不同论坛通过保存论坛论题的文件名(filename)属性区别,如程序清单 14-25 所示。

程序清单 14-25:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--forumlist.xml-->
<forums>
  <forum filename="jiangx.xml">
    <name>JSP+XML</name>
    <manager>Chen Yi</manager>
    <description>A space for discussing JSP technology</description>
  </forum>
</forums>
```

在 JSP+XML BBS 系统中,将 forumlist.xml 中的多个论坛基本信息通过 user.jsp 显示,如图 14-18 所示。user.jsp 的代码如程序清单 14-26 所示。

程序清单 14-26:

```
<!--user.jsp-->
<%@page contentType="text/html;charset=gb2312"%>
<%@taglib uri="/WEB-INF/MyTagLib" prefix="forumlist"%>
<html>
<head>
  <title>用户进入论坛主题区</title>
   <link href="resource/style.css" rel="stylesheet" type="text/css">
</head>
<body>
  <h3>JSP+XML BBS论坛系统</h3>
  <table border="1">
  <tr>
    <th>论坛</th><th>版主</th><th>简介</th>
  </tr>
  <forumlist:forums auth="user"/>    <!--自定义标签-->
  </table>
</body>
</html>
```



图 14-18　用户论坛导航页面

在 user.jsp 中定义一个自定义标签 forums，属性为 auth。forums 标签表示显示论坛列表中的所有基本论坛的信息。属性 auth 表示，根据客户的类别不同，显示效果不同。如果 auth 为 user，按照用户界面的要求显示，如果 auth 为 manager 表示按照管理员的要求界面显示。定义这样的自定义标签有标签类：ForumListTag，见程序清单 14-27。

程序清单 14-27：

```
//ForumListTag.java
package tags;
import javax.servlet.jsp.*;
```

```java
import javax.servlet.jsp.tagext.* ;
import javax.servlet.http.* ;
public class ForumListTag extends TagSupport{
    private String auth;
    public int doStartTag()throws JspException{
        try{
            ForumListBuilder builder=new ForumListBuilder(auth);                //创建论坛表实例
            HttpSession session=pageContext.getSession();
            JspWriter out=pageContext.getOut();
            String username=(String)session.getAttribute("username");
            if(username! =null)
            out.println("欢迎你: "+username);
            out.println(builder.getRecords());
        }catch(Exception e){
            throw new JspException(e.getMessage()+"出现错误");
        }
        return SKIP_BODY;
    }
    public String getAuth() {
        return auth;
    }
    public void setAuth(String auth) {
        this.auth=auth;
    }
}
```

在标签类 ForumListTag.java 中,是依赖 ForumListBuilder 类利用 SAX API 实现对论坛列表 forumlist.xml 的遍历。ForumListBuilder.java 见程序清单 14-28。

程序清单 14-28:

```java
//ForumListBuilder.java
package tags;
import javax.servlet.jsp.* ;
import javax.xml.parsers.* ;
import org.xml.sax.* ;
import org.xml.sax.helpers.DefaultHandler;
public class ForumListBuilder {
        private StringBuffer result;
        private SAXParserFactory factory;
        private SAXHandler handler;
        private String fileName;
        private String auth;                        //权限
        public ForumListBuilder(String auth){
            result=new StringBuffer("");
            fileName=" forumlist.xml";
            this.auth=auth;
```

```
    }
    public void parseToRecords()throws JspException{
        try{
            factory=SAXParserFactory.newInstance();
            SAXParser parser=factory.newSAXParser();
            XMLReader reader=parser.getXMLReader();
            handler=new SAXHandler();
            reader.setContentHandler(handler);
            reader.parse(fileName);
        }catch(Exception e){
            System.out.println(e.getMessage());
            throw new Error("警告:严重错误!"+e.getLocalizedMessage());
        }
    }
    public StringBuffer getRecords()throws JspException{
        parseToRecords();
        result.append(handler.getRecords());
        return result;
    }
    class SAXHandler extends DefaultHandler{          //用 SAX API 遍历 xml 文件
        StringBuffer record;
        StringBuffer s;
        String temp;
        String filename;
        String name;
        String manager;
        String description;
        public void startDocument()throws SAXException{
        record=new StringBuffer();
        }
      public void startElement(String namespaceURI,String localName,String qName,Attributes attrs)
      throws SAXException{                            //处理元素的开始
            s=new StringBuffer();
            if(qName.equals("forum"))
              filename=attrs.getValue("filename");
        }
        public void characters(char[] chars,int start,int length) throws SAXException{
        //处理元素的内容
          s.append(chars,start,length);
        }
public void endElement(String namespaceURI,String localName,String qName)
        throws SAXException{                          //处理元素的结束
          String text=s.toString().trim();
          if(qName.equals("name"))    name=text;
          else if(qName.equals("manager")) manager=text;
```

```
            else if(qName.equals("description")){
            description=text;
            if(auth.equals("user")){
                setUserList();
            }
            else if(auth.equals("manager")){
                setManagerList();
            }
            else return;
            }
        }
        public void setUserList(){                    //输出用户显示效果
        record.append("<tr>"+
                    "<td><a href='forum.jsp?filename="+filename+
                    "&forumName="+name.replaceAll(" ","+")+"'>"+
                    name+"</a></td>"+
                    "<td>"+manager+"</td>"+
                    "<td>"+description+"</td></tr>");
        }
        public void setManagerList(){                 //输出管理员的显示效果
        record.append("<tr>"+
                    "<td>"+name+"</a></td>"+
                    "<td>"+manager+"</td>"+
                    "<td>"+description+"</td>"+
                    "<td><form action='DeleteForumServlet' method='post'>"+
                    "<input type=\"hidden\" name=\"filename\" "+
                    "value=\""+filename+"\">"+
                    "<input type='submit' value='删除'/>"+
                    "</form></td></tr>");
        }
        public StringBuffer getRecords(){
        return record;
        }
    }//end class SAXHandler
}
```

14.4.5 用户发表新信息

用户在论坛导航界面中,单击列表中显示论坛的名称,就可以实现论坛的选择。用户选择主题论坛后,可以发表新信息。一个新信息具有的主要属性有编号(id)、标题(title)、提交者(author)、提交内容(content)、提交时间(date)和回复数(reply_time)。每一个主题实体具有多个回复。"回复"的主要属性有"回复主题"(reply_title)、"回复者"(reply_author)、"回复时间"(reply_date)、"回复内容"(reply_content)。由于论坛中的信息必须依赖于主题论坛才具有意义。保存详细论坛信息的 XML 模板如程序清单 14-29 所示。

程序清单 14-29：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<forum filename="jiangx.xml">
  <name/>
  <manager/>
  <topic id=" ">
    <title/ >
    <author/>
    <content/>
    <date/>
    <reply_time/>
    <reply>
      <reply_title/>
      <reply_author/>
      <reply_content/>
      <reply_date/>
    </reply>
</topic>
```

发表新信息的界面是由 forum.jsp 来实现的，运行结果如图 14-19 所示。forum.jsp 的代码见程序清单 14-30。

程序清单 14-30：

```
<!--forum.jsp-->
<%@page contentType="text/html; charset=UTF-8" language="java"%>
<%@taglib uri="/WEB-INF/MyTagLib" prefix="forum"%>
<html>
<head>
   <title>JSP+XML论坛</title>
    <link href="resource/style.css" rel="stylesheet" type="text/css">
</head>
<body>
   <%
    String filename=request.getParameter("filename");          //确定论坛文件
    String forumName=request.getParameter("forumName");         //确定论坛的名称
    if(filename==null)filename="";
    if(forumName==null)forumName="";
   %>
   <h3><a href="user.jsp">JSP+XML BBS论坛系统</a></h3>
   <forum:topic filename="<%=filename%>"/><!--显示指定论坛的所有主题-->
   <form action="AddTopicServlet" method="post"><!--AddTopicServlet添加新信息-->
     <table>
      <caption>我要发言</caption>
      <tr><td>主题<td></tr>
      <tr><td><input type="text" name="topic"/></td></tr>
```

```
        <tr><td>内容</td></tr>
        <tr>
            <td><textarea type="" name="content" cols="20" rows="8"></textarea></td>
        </tr>
        <tr>
            <td><input type="hidden" name="filename" value="<%=filename%>"/>
                <input type="hidden" name="forumName" value="<%=forumName%>"/>
                <input type="submit" value="发言"/>
                <input type="reset" value="重置"/>
            </td>
        </tr>
      </table>
    </form>
</body>
</html>
```



图 14-19　发表新信息的界面

在 forum.jsp 中,定义自定义标签 forum:topic 来实现对特定主题论坛中所有论题的显示。forum:topic 自定义标签的定义形式类似于上述的 forumlist:forums 标签的定义,在这里不再介绍。请用户思考自行定义。

要添加新信息到指定论坛中,是通过 AddTopicServlet 类实现的。AddTopicServlet.java 的代码见程序清单 14-31。

程序清单 14-31:

```
//AddTopicServlet.java
```

```
package servlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import java.util.*;
public class AddTopicServlet extends HttpServlet {
HttpSession session;
String filename;                                    //对应论坛的 xml 文件名
String forumName;                                   //论坛名
String title;                                       //论坛的信息标题
String author;                                      //提交信息者
String content;                                     //提交信息
    String date;                                    //提交时间
public void doGet(HttpServletRequest request,HttpServletResponse response)
    throws ServletException,IOException{
    doPost(request,response);
}
public void doPost(HttpServletRequest request,HttpServletResponse response)
    throws ServletException,IOException{
    session=request.getSession();
    RequestDispatcher  view;
    if(initAllInfo(request)){                       //初始化所有提交信息
        insertAElement();                           //将提交信息插入到指定文件中
    }
        view=request.getRequestDispatcher("forum.jsp");              //提交成功浏览主题论坛
    view.forward(request, response);
}
public boolean initAllInfo(HttpServletRequest request){                  //初始化论坛信息
    filename=request.getParameter("filename");
    forumName=request.getParameter("forumName");
    title=request.getParameter("topic");
    content=request.getParameter("content");
    author=(String)session.getAttribute("email");
    if(filename==null)return false;
    if(forumName==null)forumName="";
    if(title==null)title="";
    if(content==null)content="";
    if(author==null){author="";return false;}
    date=new Date().toString();
    return true;
```

```
}
public String getId(){                           //建立新信息的编号
      String sid;
      Date date=new Date();
      sid=forumName+date.toString();
      return sid;
}
public Node getATopic(Document document){        //创建一个新结点
      Element topicNode=document.createElement("topic");                  //创建 topic结点
      topicNode.setAttribute("id", getId());                     //设置 topic结点的属性 id

        Node titleNode=document.createElement("title");                   //创建 title结点
      titleNode.appendChild(document.createTextNode(title));
      Node authorNode=document.createElement("author");                   //创建 author结点
      authorNode.appendChild(document.createTextNode(author));
      Node contentNode=document.createElement("content");                 //创建 content结点
      contentNode.appendChild(document.createTextNode(content));
      Node dateNode=document.createElement("date");                       //创建 date结点
      dateNode.appendChild(document.createTextNode(date));
      Node replytimeNode=document.createElement("reply_time");
        //创建 reply_time结点,表示回复次数
      replytimeNode.appendChild(document.createTextNode("0"));
      topicNode.appendChild(titleNode);          //插入到 topic结点
      topicNode.appendChild(authorNode);         //插入到 topic结点
      topicNode.appendChild(contentNode);        //插入到 topic结点
      topicNode.appendChild(dateNode);           //插入到 topic结点
      topicNode.appendChild(replytimeNode);      //插入到 topic结点
      return topicNode;
}
public void insertAElement(){                    //插入结点到 xml 文件中
      DocumentBuilderFactory factory;
      DocumentBuilder builder;
      String fileName="E:/jsp/JSP_XMLForum/forumXML/"+ filename;
      try{
              factory=DocumentBuilderFactory.newInstance();
              builder=factory.newDocumentBuilder();
              Document document=builder.parse(new File(fileName));
              Element element=document.getDocumentElement();
              element.appendChild(getATopic(document));         //插入
              TransformerFactory t=TransformerFactory.newInstance();
              Transformer transformer=t.newTransformer();
              DOMSource source=new DOMSource(document);
              StreamResult result=new StreamResult(new File(fileName));
              transformer.transform(source,result);             //转换
      }catch(Exception e){
```

```
            System.out.println(e.getMessage());
        }
    }
}
```

14.4.6 用户发表回复

用户可以针对论坛的特点信息发表回复，表达自己的见解。通过单击特定信息的链接，进入 reply.jsp 回复页面。通过回复页面，提交回复信息给 AddReplyServlet，将回复信息插入到论坛对应的 xml 文件中。回复信息成功后，可以发现回复次数增加。用户发表回复信息的运行结果如图 14-20 所示。

回复页面 reply.jsp 见程序清单 14-32。

程序清单 14-32：

```
<!--reply.jsp-->
<%@page contentType="text/html; charset=UTF-8" language="java"%>
<%@taglib uri="/WEB-INF/MyTagLib" prefix="forum"%>
<html>
<head>
   <title>JSP+XML 论坛</title>
   <link href="resource/style.css" rel="stylesheet" type="text/css">
</head>
  <%    String filename=request.getParameter("filename");
        String id=request.getParameter("id");
        String title=request.getParameter("title");
  %>
   <body>
   <center>
   <h3><a href="user.jsp">JSP+XML BBS 论坛系统</a></h3>
   <forum:reply filename="<%=filename%>" id="<%=id%>"/><!--自定义标签-->
   <form action="AddReplyServlet" method="get">
      <table>
       <caption>我要发言</caption>
       <tr><td>主题<td>
       </tr>
       <tr><td><input type="text" name="reply_title"/></td>
       </tr>
       <tr><td>内容</td>
       </tr>
       <tr><td>
       <textarea type="" name="reply_content" cols="20" rows="8"></textarea>
      </td>
       </tr>
       <tr><td><input type="hidden" name="filename" value="<%=filename%>"/>
              <input type="hidden" name="id" value="<%=id%>"/>
```

```
                <input type="submit" value="发言"/>
                <input type="reset" value="重置"/>
            </td>
        </tr>
      </table>
    </form>
    </center>
</body>
</html>
```



图 14-20　用户回复信息界面

reply.jsp 中利用自定义标签<forum:reply>，按照标签指定属性 filename 的值，将指定的论坛对应的 XML 文件显示出来。请用户编写代码实现上述功能。

处理回复信息的 AddReplyServlet 的源代码见程序清单 14-33。

程序清单 14-33：

```
//AddReplyServlet.java
package servlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
```

```java
import java.util.* ;
public class AddReplyServlet extends HttpServlet {
    HttpSession session;
    String filename;                                   //文件名
    String id;                                         //指定编号
    String author;                                     //指定提交者
    String reply_title;                                //指定回复标题
    String reply_content;                              //指定回复内容
    String date;                                       //指定回复时间
    public void doGet(HttpServletRequest request,HttpServletResponse response)
      throws ServletException,IOException{
        doPost(request,response);
    }
    public void doPost(HttpServletRequest request,HttpServletResponse response)
    throws ServletException,IOException{
        session=request.getSession();
        RequestDispatcher   view;
        if(initAllInfo(request)){
            insertAElement();
        }
            view=request.getRequestDispatcher("reply.jsp");
            view.forward(request, response);
    }
    public boolean initAllInfo(HttpServletRequest request){
        filename=request.getParameter("filename");
        id=request.getParameter("id");
        reply_title=request.getParameter("reply_title");
        reply_content=request.getParameter("reply_content");
        author=(String)session.getAttribute("email");
        if(id==null)return false;
        if(filename==null)     return false;
        if(reply_title==null)      reply_title="";
        if(reply_content==null)     reply_content=" ";
        if(author==null){
            author="";
            return false;
            }
        date=new Date().toString();
        return true;
    }
    public Node location(String id,Document document){            //查找指定编号的信息
        NodeList list=document.getElementsByTagName("topic");
        for(int i=0;i<list.getLength();i++){
            Node node=list.item(i);
            Node attri=node.getAttributes().getNamedItem("id");
```

```
            if(attri!=null){
                if(attri.getNodeValue().equals(id)){
                    return node;
                }
            }
        }
        return null;
    }
    public Node createAReply(Document document){                //新建 reply 结点
        Element topicNode=document.createElement("reply");
        Node titleNode=document.createElement("reply_title");
        titleNode.appendChild(document.createTextNode(reply_title));
        Node authorNode=document.createElement("reply_author");
        authorNode.appendChild(document.createTextNode(author));
        Node contentNode=document.createElement("reply_content");
        contentNode.appendChild(document.createTextNode(reply_content));
        Node dateNode=document.createElement("reply_date");
        dateNode.appendChild(document.createTextNode(date));
        topicNode.appendChild(titleNode);
        topicNode.appendChild(authorNode);
        topicNode.appendChild(contentNode);
        topicNode.appendChild(dateNode);
        return topicNode;
    }
    public void changeReplyTimes(Document document){            //改变 node 的 reply_time 次数
        int i=0;
        Node newNode=null;
        Node oldNode=null;
        String time;
        Node node=location(id,document);                        //查找指定 topic;
        if(node==null) return;
            oldNode=node.getFirstChild();
        while(oldNode!=null){
            if(oldNode.getNodeName().equals("reply_time")){
                time=oldNode.getTextContent();
                try{
                  i=Integer.parseInt(time);
                  i++;
                  time=String.valueOf(i);
                }catch(Exception e){
                    System.out.println(e.getMessage());
                }
                newNode=document.createElement("reply_time");
                newNode.appendChild(document.createTextNode(time));
                node.replaceChild(newNode, oldNode);
```

```
                    break;
                }
                oldNode=oldNode.getNextSibling();
            }
        }
        public boolean insertAElement(){
            DocumentBuilderFactory factory;
            DocumentBuilder builder;
            String fileName=filename;
            try{
                factory=DocumentBuilderFactory.newInstance();
                builder=factory.newDocumentBuilder();
                Document document=builder.parse(new File(fileName));
                Element element=(Element)location(id,document);            //查找指定 topic;
                if(element==null) return false;
                element.appendChild(createAReply(document));
                Element e=document.getDocumentElement();
                e.appendChild(element);
                changeReplyTimes(document);            //修改 topic的 reply_time值

                TransformerFactory t=TransformerFactory.newInstance();
                Transformer transformer=t.newTransformer();
                DOMSource source=new DOMSource(document);
                StreamResult result=new StreamResult(new File(fileName));
                transformer.transform(source,result);
            }catch(Exception e){
                System.out.println(e.getMessage());
            }
            return true;
        }
    }
```

14.4.7 管理员的论坛管理

管理员的主要智能是管理和维护论坛系统，如图 14-21 所示。在 JSP+XML BBS 论坛系统中，管理员可以实现对论坛的管理，具体涉及浏览论坛信息、增加新的论坛、删除论坛等。

1. 增加新论坛

管理员可以增加新的论坛。设置论坛的名称，指派用户级别为“高”的用户作为论坛的版主。在 addForum.jsp（见程序清单 14-34）中定义了增加新论坛的信息提交界面，如图 14-22 所示。

程序清单 14-34：

```
<!--addForum.jsp-->
```



图 14-21　论坛管理界面

```
<%@page contentType="text/html; charset=UTF-8" language="java"%>
<%@taglib uri="/WEB-INF/MyTagLib" prefix="userlist"%>
<html>
<head>
    <title>JSP+XML论坛</title>
    <link href="resource/style.css" rel="stylesheet" type="text/css">
</head>
<body>
    <h3>JSP+XML BBS论坛管理系统</h3>
    <h4>添加论坛</h4>
    <form action="AddForumServlet" method="get">
      <table>
        <tr><td>论坛名称<input type="text" name="forum"/></td>
        </tr>
        <tr><td>指定文件<input type="text" name="filename"/></td>
        </tr>
        <tr><td>版主<userlist:users/><!--显示用户级别为"高"的用户列表-->
            </td>
        </tr>
        <tr><td>论坛简述<br>
                <textarea name="description" rows="5" rows="40"></textarea>
            </td>
        </tr>
        <tr><td><input type="submit" value="新增论坛"/></td>
        </tr>
      </table>
    </form>
</body>
</html>
```

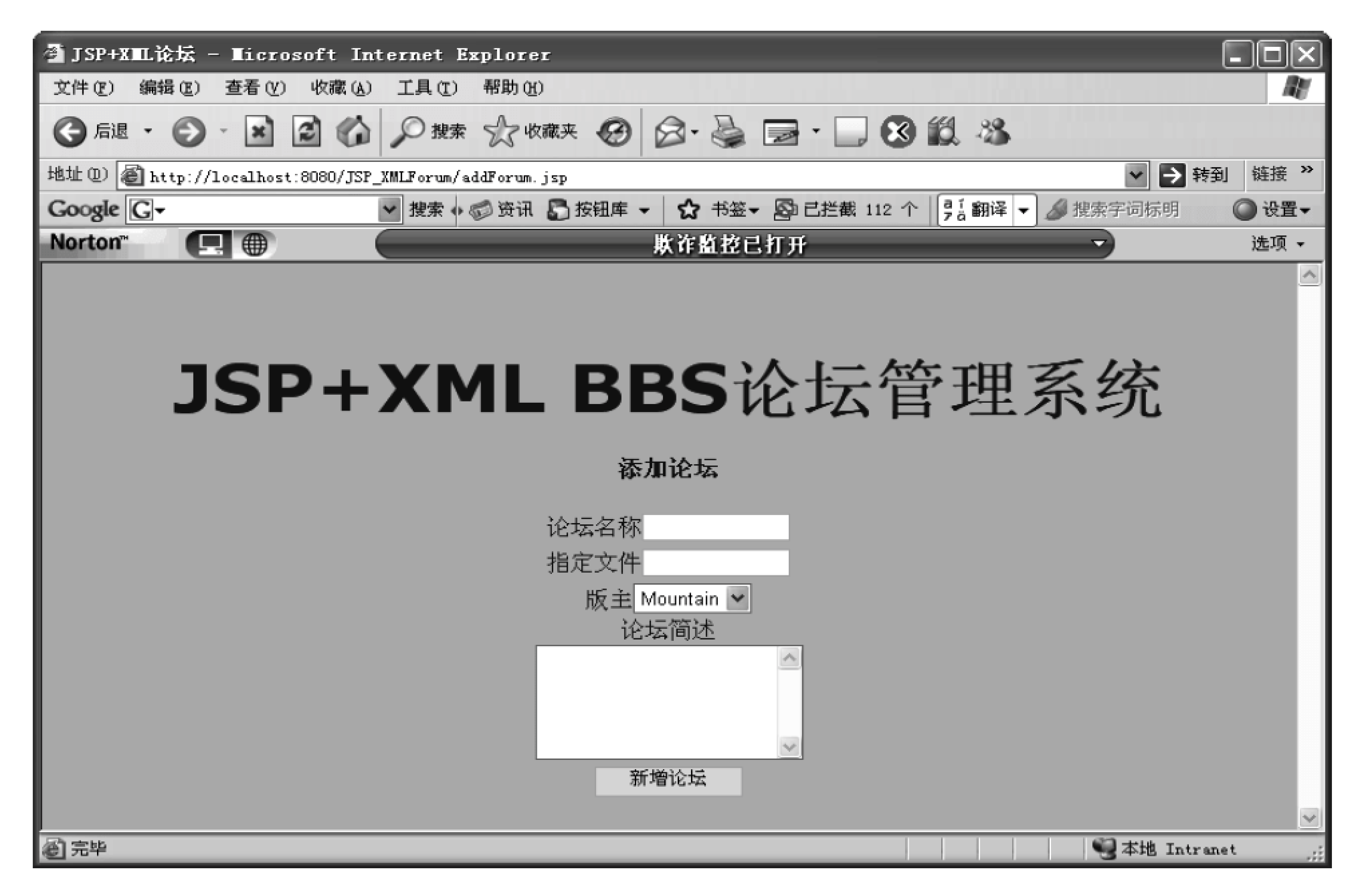


图 14-22　增加新论坛

为了显示用户级别列表为“高”的所有用户，通过自定义标签 userlist:users 来实现。该自定义标签对应的标签处理类为 UserTag。UserTag.java 是利用 DOM API 实现对 userlist.xml 文件的查询，具体内容见程序清单 14-35。

程序清单 14-35:

```
//UserTag.java
package tags;
import java.io.File;
import javax.servlet.jsp.* ;
import javax.servlet.jsp.tagext.* ;
import org.w3c.dom.* ;
import javax.xml.parsers.* ;
public class UserTag extends TagSupport{                    //自定义标签处理类
    public int doStartTag()throws JspException{
        try{JspWriter out=pageContext.getOut();
            out.println(parse());                           //输出解析内容
        }catch(Exception e){
            throw new JspException(e.getMessage()+"出现错误");
        }
        return SKIP_BODY;
    }
    public StringBuffer parse(){                            //解析 userlist.xml,显示所有级别为高的用户
        DocumentBuilderFactory factory;
        DocumentBuilder builder;
        String fileName="userlist.xml";
```

```
        StringBuffer result=new StringBuffer();
        try{
            factory=DocumentBuilderFactory.newInstance();
            builder=factory.newDocumentBuilder();
            Document document=builder.parse(new File(fileName));
            NodeList list=document.getElementsByTagName("user");
                                                        //获取 user 结点集
            result.append("<select name='manager'>");   //定义组合框
            Node temp=null;
            for(int i=0;i<list.getLength();i++){
                        Node node=list.item(i).getFirstChild();
                        while(node!=null){
                             if(node.getNodeName().equals("name")){
                               temp=node;                   //查询结点名为 name 的结点
                             }
                             if(node.getNodeName().equals("level")){                //按用户级别检索
                                  if(node.getTextContent().equals("高")){              //级别高,处理
                             result.append("<option>"+temp.getTextContent()+"</option>");
                                       temp=null;
                                  }
                             }
                             node=node.getNextSibling();
                        }
                }
                 result.append("</select>");
        }catch(Exception e){
             System.out.println(e.getMessage());
        }
            return result;
    }
    }
```

AddForumServlet 类为了实现将 addForum.jsp 提交的论坛信息添加到 forumlist.jsp,并同时创建一个管理员指定保存该论坛的 xml 文件。AddForumServlet.java 对应的程序清单如 14-36 所示。

程序清单 14-36:

```
//AddForumServlet.java
package servlet;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
```

```java
import javax.xml.transform.stream.* ;
public class AddForumServlet extends HttpServlet {              //增加新论坛
     String name;                                                //论坛名
     String filename;                                                     //保存论坛所有信息的 xml 文件
     String manager;                                             //版主
     String description;                                         //论坛的简单介绍
     File file;
     public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException{
          doPost(request,response);
     }
     public void doPost(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException{
          if(initInfo(request)){                                 //初始化提交信息
               insertAForum();                                   //插入新论坛
               createAForumFile(filename);                                //创建保留论坛信息的 xml 文件
          }
          RequestDispatcher view= request.getRequestDispatcher("manager.jsp");
     view.forward(request, response);
     }
     public boolean initInfo(HttpServletRequest request){        //获取提交信息
          name= request.getParameter("forum");
          filename= request.getParameter("filename");
          manager= request.getParameter("manager");
          description= request.getParameter("description");
          if(name==null) return false;
          if(filename==null)     return false;
          if(manager==null)manager= "管理员";
          if(description==null)description= "";
          return true;
     }
     public Node createAForumRecord(Document document){          //创建一个新结点
          Element forumNode= document.createElement("forum");
          forumNode.setAttribute("filename", filename);
          Node nameNode= document.createElement("name");         //论坛名
          nameNode.appendChild(document.createTextNode(name));
          Node managerNode= document.createElement("manager");        //版主
          managerNode.appendChild(document.createTextNode(manager));
          Node descNode= document.createElement("description");       //描述
          descNode.appendChild(document.createTextNode(description));
                    forumNode.appendChild(nameNode);
          forumNode.appendChild(managerNode);
          forumNode.appendChild(descNode);
          return forumNode;
     }
```

```java
    public void createAForumFile(String filename){                //创建保存论坛信息的 xml 文件
    try{
    File file=new File("forumXML",filename);                     //创建 forumXML 目录下的 filename 文件
    if(file.exists()) return;
    file.createNewFile();
    FileOutputStream stream=new FileOutputStream(file);
    OutputStreamWriter ostream=new OutputStreamWriter(stream);
    ostream.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");       //写入论坛的基本信息
    ostream.write("<forum filename=\""+filename+"\">");
    ostream.write("<name>"+name+"</name>");
    ostream.write("<manager>"+manager+"</manager>");
    ostream.write("</forum>");
    ostream.close();
    stream.close();
    }catch(Exception e){
        System.out.println(e.getMessage());
    }
    }
    public void insertAForum(){                  //插入新论坛基本内容到论坛列表 forumlist.xml 中
        DocumentBuilderFactory factory;
        DocumentBuilder builder;
        String fileName=" forumlist.xml";
        try{
                factory=DocumentBuilderFactory.newInstance();
                builder=factory.newDocumentBuilder();
                Document document=builder.parse(new File(fileName));
                Element element=document.getDocumentElement();
                element.appendChild(createAForumRecord(document));
                TransformerFactory t=TransformerFactory.newInstance();
                Transformer transformer=t.newTransformer();
                DOMSource source=new DOMSource(document);
                StreamResult result=new StreamResult(new File(fileName));
                transformer.transform(source,result);
        }catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```

2. 删除论坛

对于一些无意义的论坛可以给予删除。提供删除论坛的基本信息是通过 deleteForum.jsp 实现，界面如图 14-23 所示。具体的删除处理是通过 DeleteForumServlet 类完成。

提交删除论坛基本信息的 deleteForum.jsp 的代码见程序清单 14-37。

程序清单 14-37：



图 14-23　删除论坛的界面

```
<!--deleteForum.jsp-->
<%@page contentType="text/html; charset=UTF-8" language="java"%>
<%@taglib uri="/WEB-INF/MyTagLib" prefix="forumlist"%>
<html>
<head>
    <title>JSP+XML论坛</title>
    <link href="resource/style.css" rel="stylesheet" type="text/css">
</head>
<body>
    <h3>JSP+XML BBS论坛管理系统</h3>
    <h4>删除论坛</h4>
    <table border="1">
        <forumlist:forums auth="manager"/><!--见 14.4.4论坛导航-->
    </table>
</body>
</html>
```

具体实现删除论坛包含两方面内容：一方面需要删除论坛中发表的所有的信息和回复，即将保留信息的 xml 文件删除；另一方面是从论坛列表 forumlist.xml 中将关于删除论坛的基本记录删除。用 DeleteForumServlet 实现论坛删除。DeleteForumServlet.java 的代码见程序清单 14-38。

程序清单 14-38：

```
//DeleteForumServlet.java
package servlet;
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
public class DeleteForumServlet extends HttpServlet   {
    String filename;                                    //删除论坛文件名
    public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException{
        doPost(request,response);
    }
    public void doPost(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException{
        filename=request.getParameter("filename");
        File file=new File("forumXML",filename);
        if(file.exists())    file.delete();                 //删除保留论坛信息的 xml 文件
        deleteAForum(filename);                      //从 forumlist.xml 中删除论坛基本信息
        RequestDispatcher view=request.getRequestDispatcher("manager.jsp");
        view.forward(request, response);
    }
    public void deleteAForum(String filename){               //从 forumlist.xml 中删除论坛基本信息
        DocumentBuilderFactory factory;
        DocumentBuilder builder;
        String fileName=" forumlist.xml";
        try{    factory=DocumentBuilderFactory.newInstance();
           builder=factory.newDocumentBuilder();
           Document document=builder.parse(new File(fileName));
           NodeList list=document.getElementsByTagName("forum");               //获取 forum 集
           for(int i=0;i<list.getLength();i++){
               Node node=list.item(i);
               Node attri=node.getAttributes().getNamedItem("filename");
               if(attri.getTextContent().equals(filename)){          //检索指定文件名的结点
                   node.getParentNode().removeChild(node);           //从父结点中删除该结点
                   break;
               }
           }
           TransformerFactory t=TransformerFactory.newInstance();
           Transformer transformer=t.newTransformer();
           DOMSource source=new DOMSource(document);
           StreamResult result=new StreamResult(new File(fileName));
           transformer.transform(source,result);
        }catch(Exception e){
         System.out.println(e.getMessage());
```

```
        }
    }
}
```

小　结

JSP与XML结合可以开发功能强大的Web应用。JSP可以通过设置内容类型直接生成各类型的XML文件,包括XML文件本身、XHTML和WML等。也可以结合其他技术生成XML文件,如JavaBean。在实际应用中需要对XML文件解析,然后执行相应处理。JSP可以通过JAXP API的Java应用接口实现对XML文件的解析。通常的解析方式有DOM方式和SAX方式。这两种方式各有特色。DOM解析方式需要将XML文件全部导入执行效率不如SAX方式。但是DOM方式方便实现对XML文件删除、增加、修改结点等操作,处理方式灵活。JSP和XML相互作用的最大效果是实现应用。在具体应用XML体现在3方面:JSP用JavaBean封装XML数据,为处理提供方便;JSP中可以利用自定义标签封装XML数据;JSP可以实现利用特定的XSLT转换XML文件,方便客户在不同浏览器下应用自如。

为了说明和了解JSP和XML开发Web应用的特点,本章介绍了一个完全用JSP+XML开发的简单坛系统应用实例,来帮助用户理解JSP与XML的结合应用。

习　题　14

1. 填空题

(1) JSP直接生成XML文件需要设置<%@page%>指令的________属性值为________。

(2) JSP直接生成WML文件需要设置<%@page%>指令的________属性为________。

(3) DOM是________。

(4) SAX的简称是________。

(5) JSP转换XML文件,根据导入XML数据的方式有________、________、________和________。

2. 选择题

(1) 开发自定义标签的正确步骤序列是________。

① 编写处理标签Java类;

② 用XML文件保存标签库的描述信息;

③ 在JSP文件导入标签库,以及应用标签库的标签。

A. ③②①　　B. ①②③　　C. ②①③　　D. 以上序列均不正确

(2) 下列代码的功能是用指定的XSLT转换XML文件,代码存在错误的是________。

```
第 1行: <%@ taglib uri="http://jakarta.apache.org/taglibs/xsl-1.0" prefix="xsltr"%>
第 2行: <html>
```

第 3行： <body>
第 4行： <center>
第 5行： <h3>XSLT转换 XML文件</h3>
第 6行： <xsltr:import xsl="/XSLT5-16.xsl">
第 7行： <xsltr:include page="/XML5-16.xml"/>
第 8行： </xsltr:import>
第 9行： <hr/>
第 10行： </center>
第 11行： </body>
第 12行： </html>

A. 第 1 行　　B. 第 6 行　　C. 第 7 行　　D. 第 8 行

3. 实验题

用 JSP+XML 开发一个留言板。

附录 A　Eclipse 与 Tomcat 的整合及使用

A.1　Eclipse 和 MyEclipse 的安装

Eclipse 是当前主流集成开发环境(IDE)之一，最初是 IBM 公司研发的软件产品。目前，IBM 已经将 Eclipse 捐献给开放源代码 Eclipse.org 组织。Eclipse 开发环境基于 Java 平台，具有独创的可扩展的开发平台特性，这吸引了众多企业加入到 Eclipse 联盟中。

但是，单有 Eclipse 环境是无法开发 Web 应用的，只能作为 Java 的 IDE 来使用。所以必须使用其他支持 Web 应用的插件。MyEclipse 是 Genuitec 公司发布的支持 J2EE(Java platform 2，Enterprise Edition)的 Eclipse 插件。到 2007 年 12 月初为止，Genuitec 公司发行的 MyEclipse 的版本是 MyEclipse 6.0.1GA，该版本有两种形式。

(1) ALL-IN-ONE 形式，即将 MyEclipse 6.0.1 和 Eclipse 3.3 整合在一起，只需要将安装包安装指定顺序进行操作就可以；

(2) PLUG-IN 形式，只单独提供 MyEclipse 6.0.1 GA。

对于第二种形式，安装时，需要先安装 Eclipse，然后在将 MyEclipse 插入到 Eclipse 中。在这里，将对这种形式详细介绍。

在安装 Eclipse 和 MyEclipse 之前，需要确保安装和配置了 JDK 以及 Tomcat 服务器，具体的操作步骤见第 8 章。

1. Eclipse 的下载

可以从站点 http://www.eclipse.org/downloads/中下载 Eclipse 最新版本。在本书采用的是 Eclipse 3.3 的开发平台。

2. 安装 Eclipse 平台

Eclipse 的安装非常简单，只需要将下载的软件包解压在安装路径，如 D:\Eclipse 3.3 下就可。这时，Eclipse 的所有文件就会安装在 D:\Eclipse 3.3\eclipe 目录。

3. 安装 Eclipse 的语言包

如果需要 Eclipse 可以运行在中文环境中方便运用。可以从站点 http://update.eclipse.org/downloads/下载语言包，如 NLpack1-eclipse-SDK-3.3-win32.zip。值得注意的是，采用的语言包则必须和 Eclipse 的版本相同，否则会导致 Eclipse 平台中部分内容不能正常显示或语言包失效。由于 Eclipse 3.3 还没有对应的语言包，所以 Eclipse SDK 部分内容没有显示中文。

安装的过程如下。

首先，将语言压缩包解压在某个路径中，假设 D:\language，那么解压后的语言包放在 D:\language\eclipse 中。

然后，在 Eclipse 安装目录新建一个目录，命名为 language。将语言包中的 eclipse 复制

到 language 下。这时,D:\Eclipse 3.3\eclipse\language\eclipse 目录下就具有了语言包的两个目录 features 和 plugins 以及它们里的所有文件。

再次,在 Eclipse 安装路径中新建一个 links 目录(如 D:\Eclipse 3.3\eclipse\links)。在该目录下建立一个文本类型的 link 文件。假设文件名为 language.link,为该文件书写如下内容,实现对语言包的连接引用。

```
path=D:/Eclipse 3.3/eclipse/language
```

4. MyEclipse 的安装

首先,从网站 http://www.myeclipseide.com/index.php? name=Downloads&req=viewsdownload&sid=24 中下载 MyEclipse 的 PLUG-IN 版本。

本书采用 MyEclipse 解压 MyEclipse 的压缩包并重命名,假设解压后重命名为 D:\myEclipses,该压缩包中有 eclipse 目录,见图 A-1。该 eclipse 目录中包含 features 和 plugins 目录。可以分别将 D:\MyEclipse\eclipse 中的 features 和 plugins 目录下的所有文件包复制到 Eclipse SDK 安装目录下同名目录中。

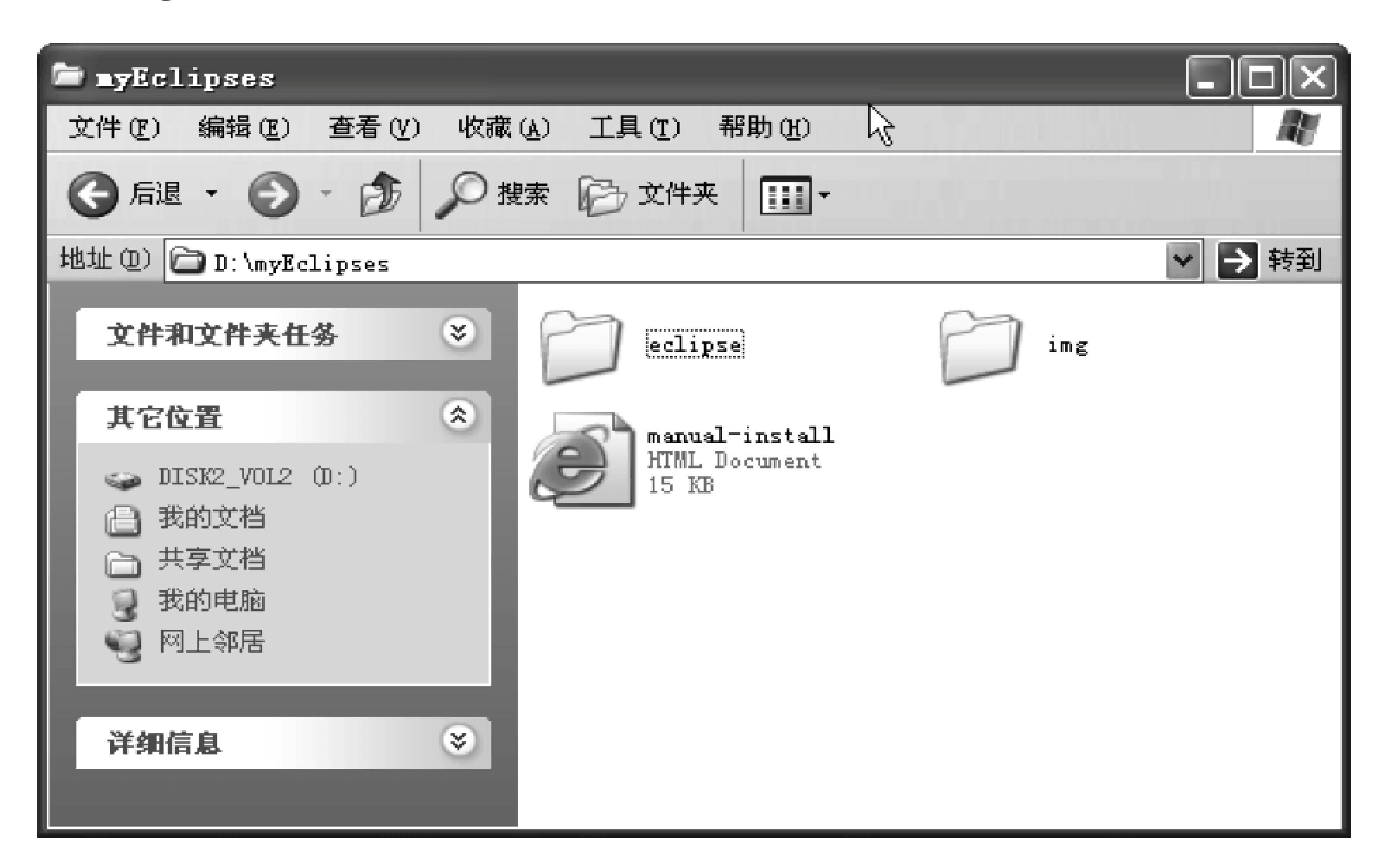


图 A-1 MyEclipse 的安装路径

也可以通过 Eclipse SDK 软件更新的方式实现 MyEclipse 6.0 GA 的安装。具体步骤如下:要在 Eclipse 设置 MyEclipse,首先启动 Eclipse,选择"帮助"|"软件更新"|"管理配置"菜单命令,进入"产品配置"窗口,见图 A-2。

在"产品配置"窗口中,在"Eclipse SDK"中右击鼠标,会弹出一个菜单。在该菜单中选择"添加"|"扩展位置",如图 A-3(a)所示。

然后如图 A-3(b)所示,选定 MyEclipse 所在的目录,如"D:\myEclipse"。设置完毕,重新启动 Eclipse,再次启动的 Eclipse 就会包含 MyEclipse 插件,如图 A-4 所示。

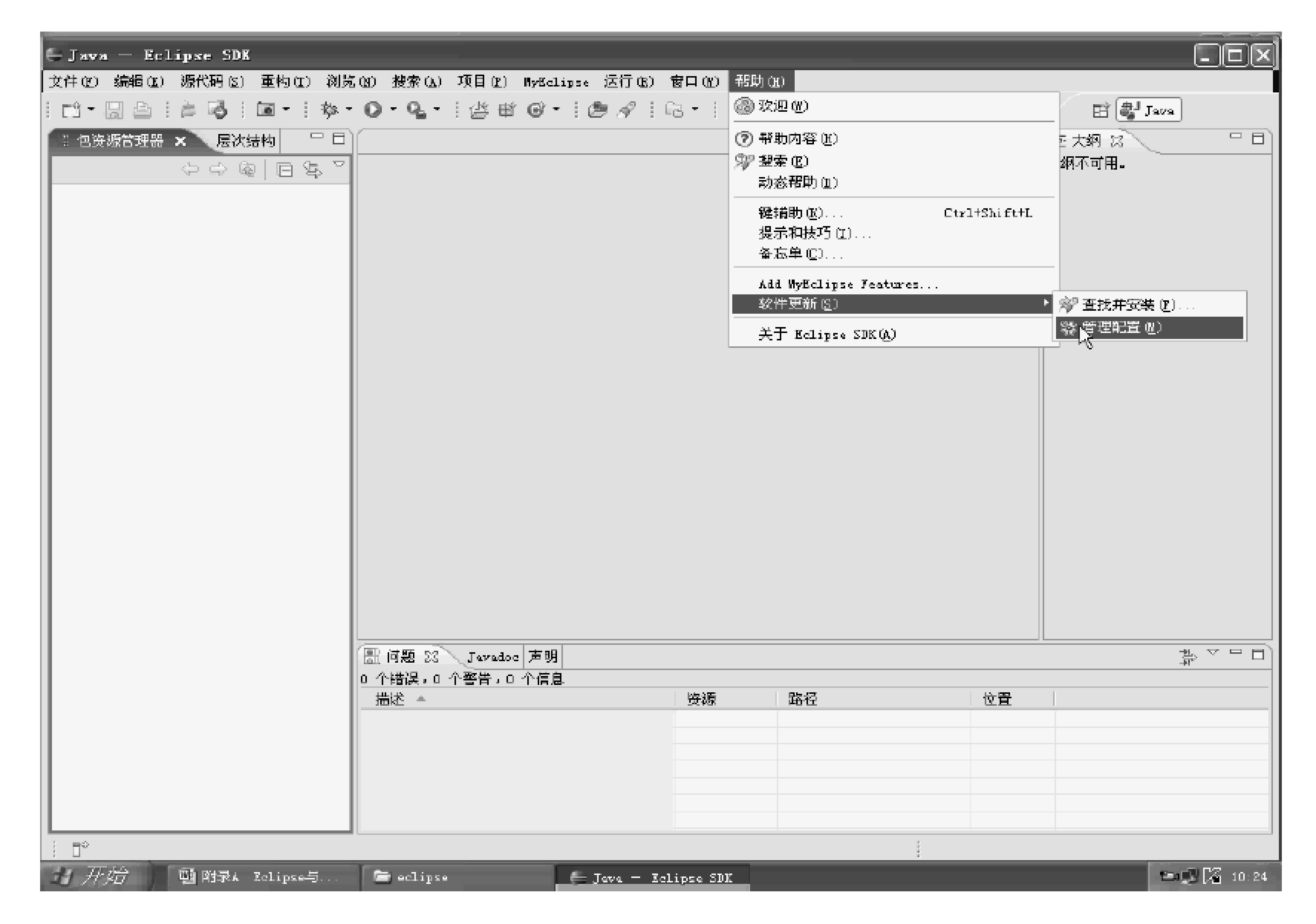


图 A-2　配置管理



(a)



(b)

图 A-3　添加插件



图 A-4　重新启动 Eclipse

A.2　Eclipse 与 Tomcat 的整合

为了让 Tomcat 和 Eclipse 整合，需要一个 Eclipse Tomcat 装载插件（Tomcat Launcher Plugin），该插件本身不具有 Tomcat 软件包，只是通过它在 Eclipse 启动 Tomcat 服务器。所以，在使用该插件之前，请确保安装和配置 Tomcat 成功。该 Eclipse Tomcat 装载插件可以到网站 http://www.eclipsetotale.com/tomcatPlugin.html 下载。

由于在编辑该书时，Eclipse Tomcat 装载插件只有 3.2 版本，本书就以 Eclipse Tomcat Launcher Plugin Version 3.2 为例来说明。步骤如下：

（1）将从网站下载的压缩包 tomcatPluginV32.zip 解压。

（2）将解压的文件包 com.sysdeo.eclipse.tomcat_3.2 整个移动到 Eclipse 安装目录的 plugins 目录即可，如图 A-5 所示。

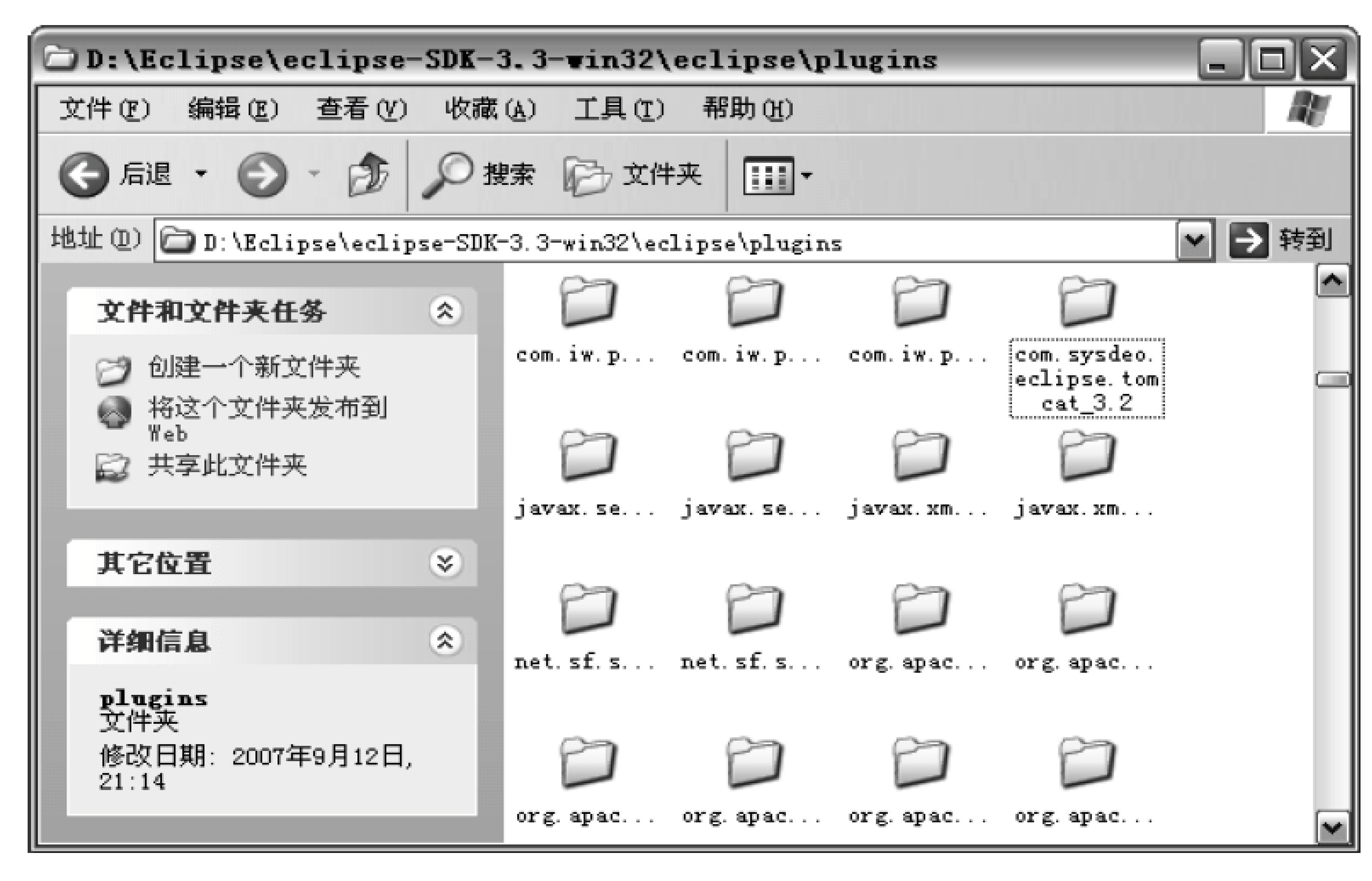


图 A-5　复制 com.sysdeo.eclipse.tomcat_3.2 到 Eclipse SDK 的 plugins 子目录

（3）重新启动 Eclipse，会在 Eclipse SDK 的菜单上出现 Tomcat 的菜单项，以及在 Eclipse SDK 的工具栏中出现 Tomcat 的 log 标记，见图 A-6。

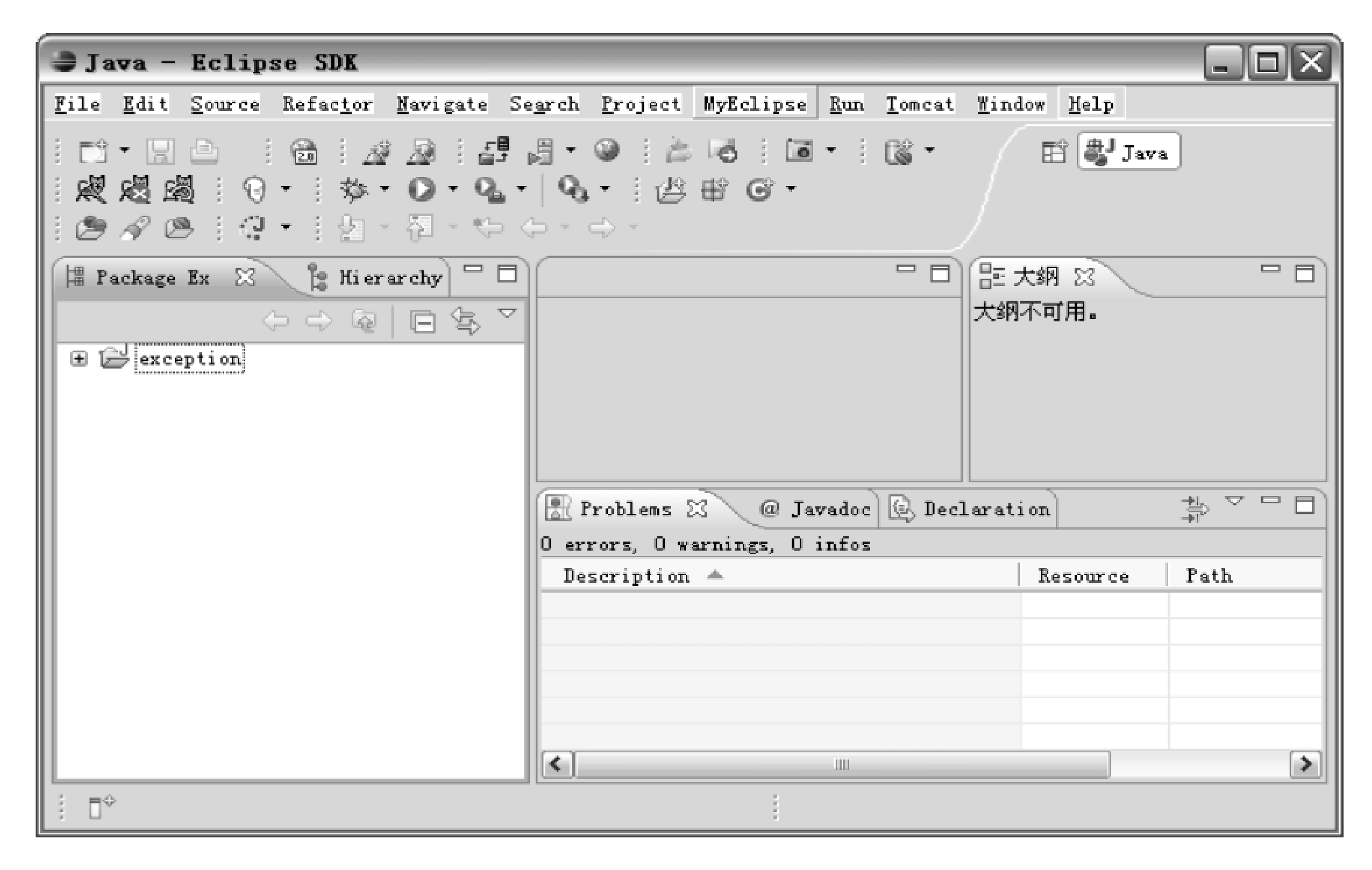
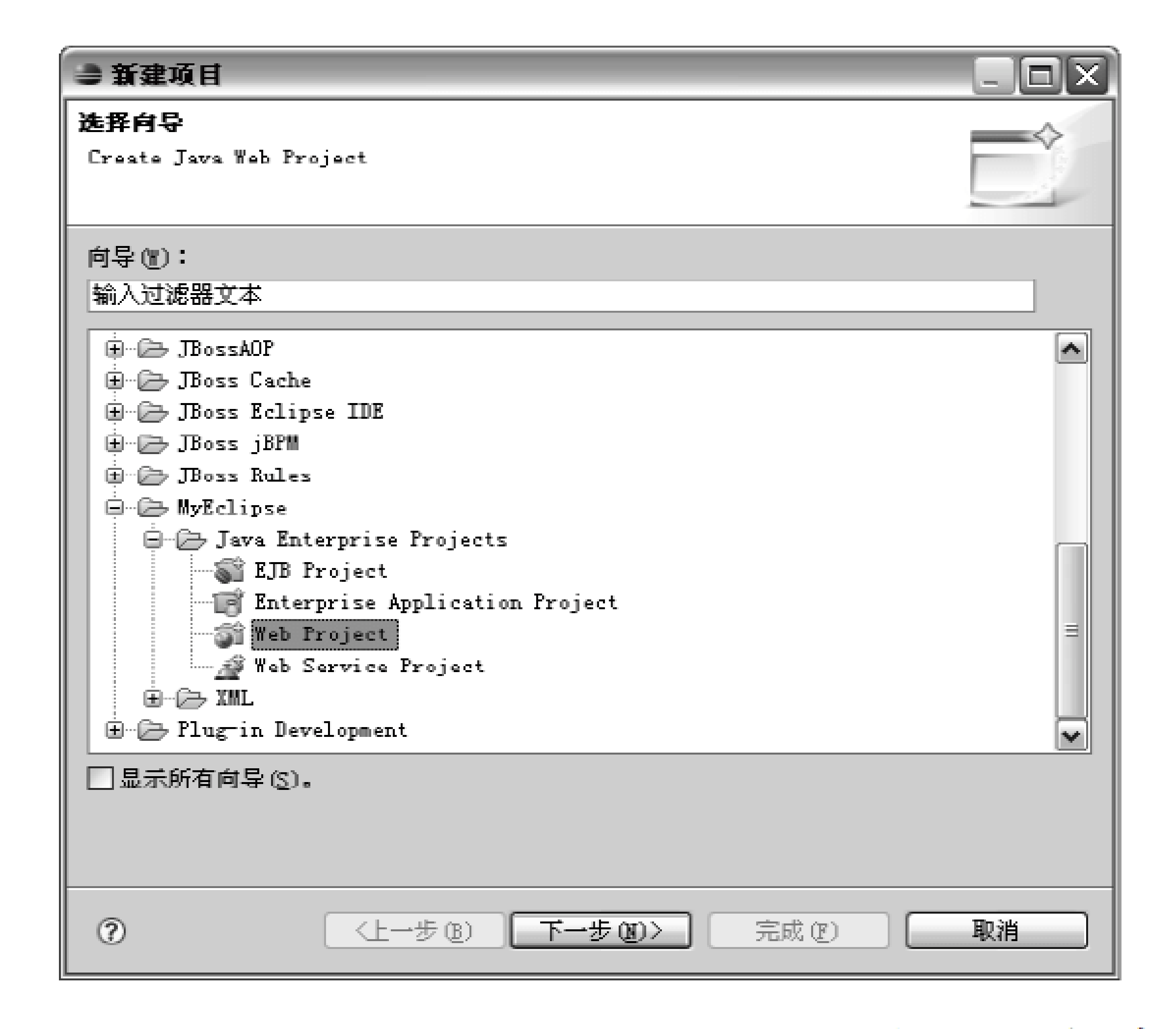


图 A-6　安装 Eclipse Tomcat 装载插件成功

A.3　Eclipse 开发一个 Web 应用

现在使用整合后的 Eclipse SDK 建立一个 Web 应用。首先必须在 Eclipse SDK 新建一个 Web 项目，即在“新建项目”的“选择向导”中选择“MyEclipse”项中“Web Project”子项，见图 A-7。



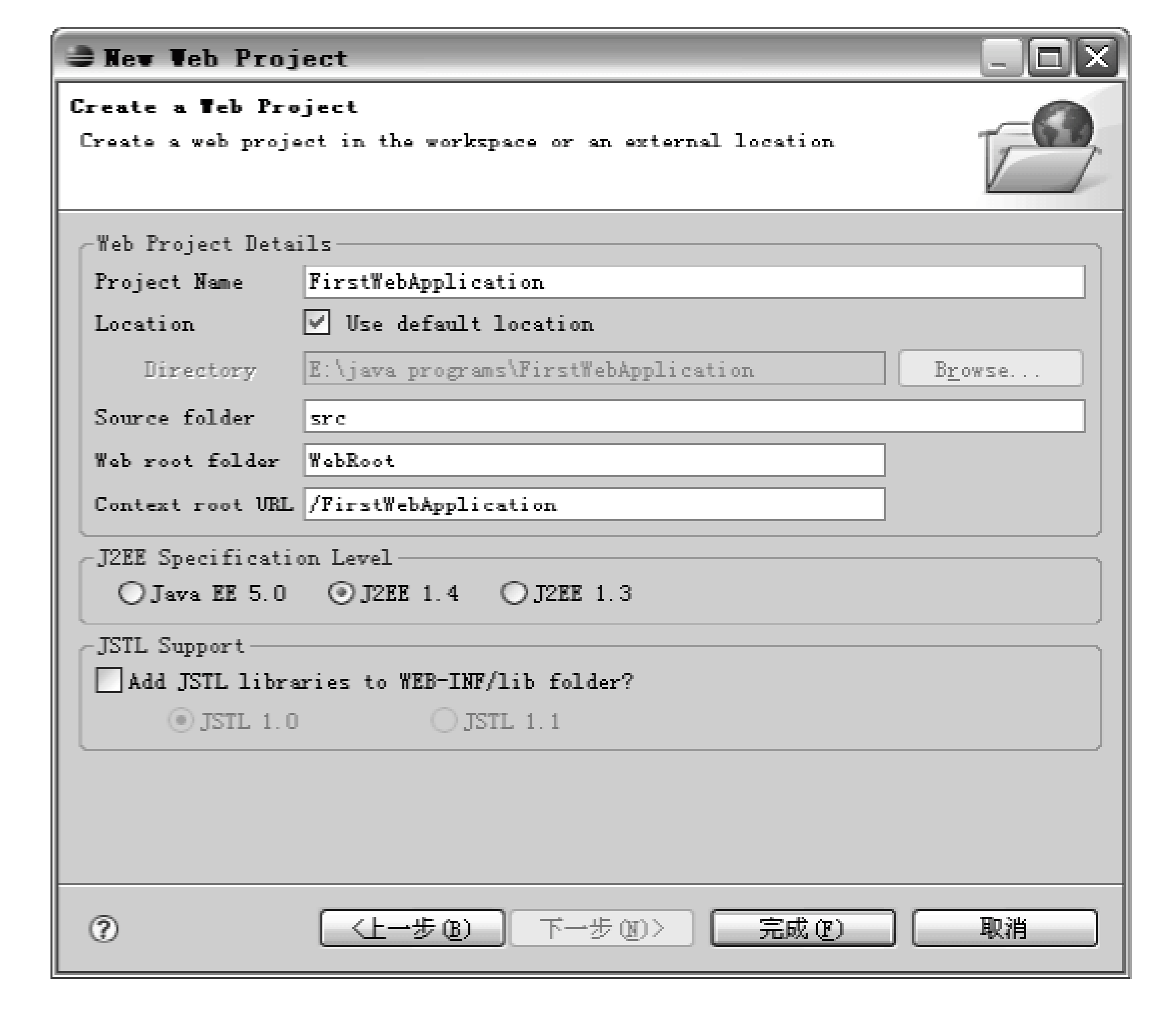


图 A-7　新建一个 Web 项目

假设需要建立一个 FirstWebApplication 的新项目，则在随后出现的 New Web Project（新建 Web 项目）中创建一个项目名为 FirstWebApplication 的新项目，依次设定 Source folder（源文件夹）、Web root folder（Web 根文件夹）以及 Context root URL（上下文根

URL)，然后单击“完成”按钮即可。

这时，在 Eclipse SDK 右边的 Package Explorer（包浏览）中会出现 FirstWebApplication 项目的相关文件包和文件目录。然后在 src 中右击，会弹出一个菜单。选择“新建”的 JSP 项目，如图 A-8 所示。



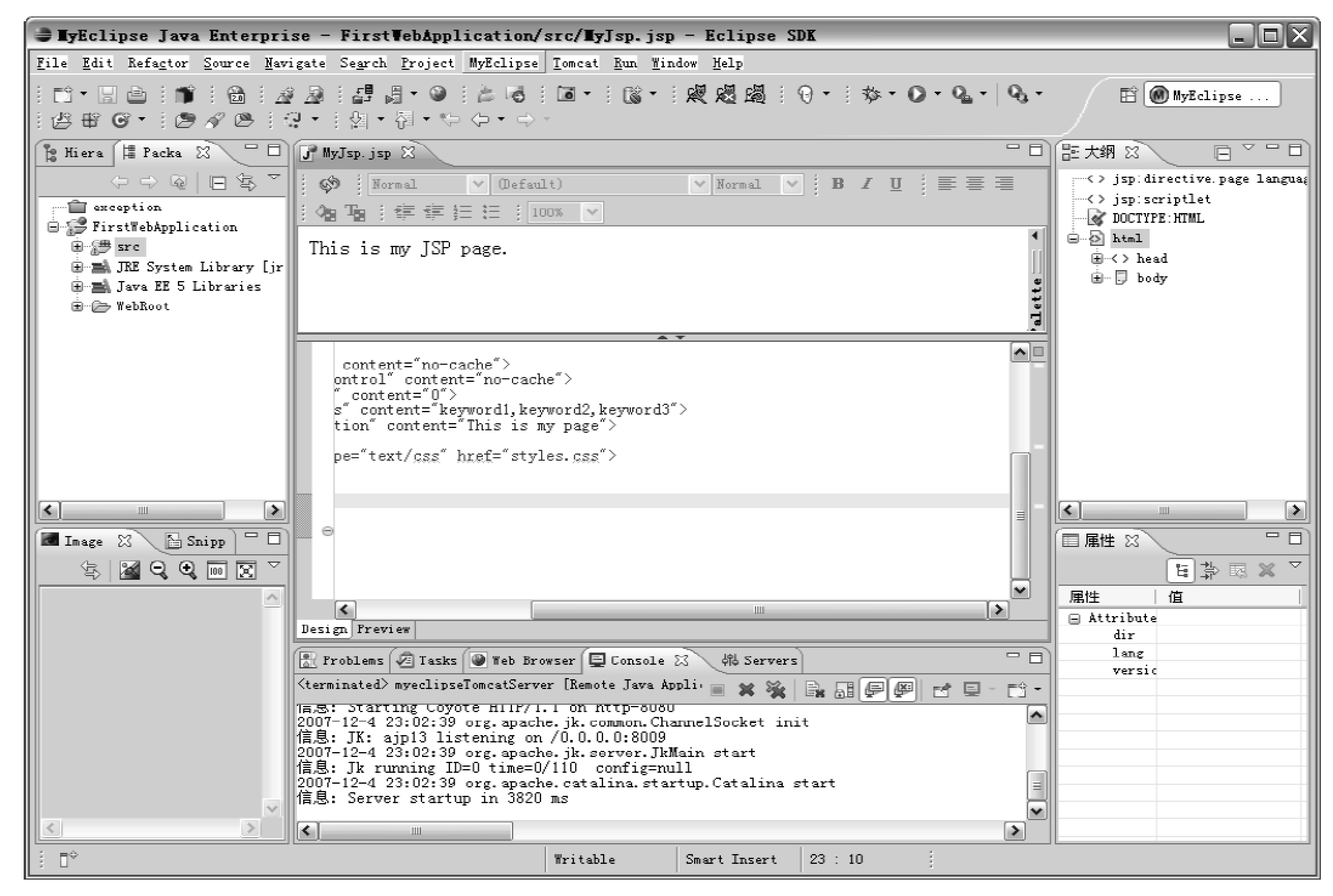


图 A-8 新建文件

为了测试该 jsp 文件是否可以运行，可以选择 Eclipse SDK 中的 Run 菜单项。Eclipse SDK 会启动 Tomcat，运行该 jsp 文件。运行结果见图 A-9。

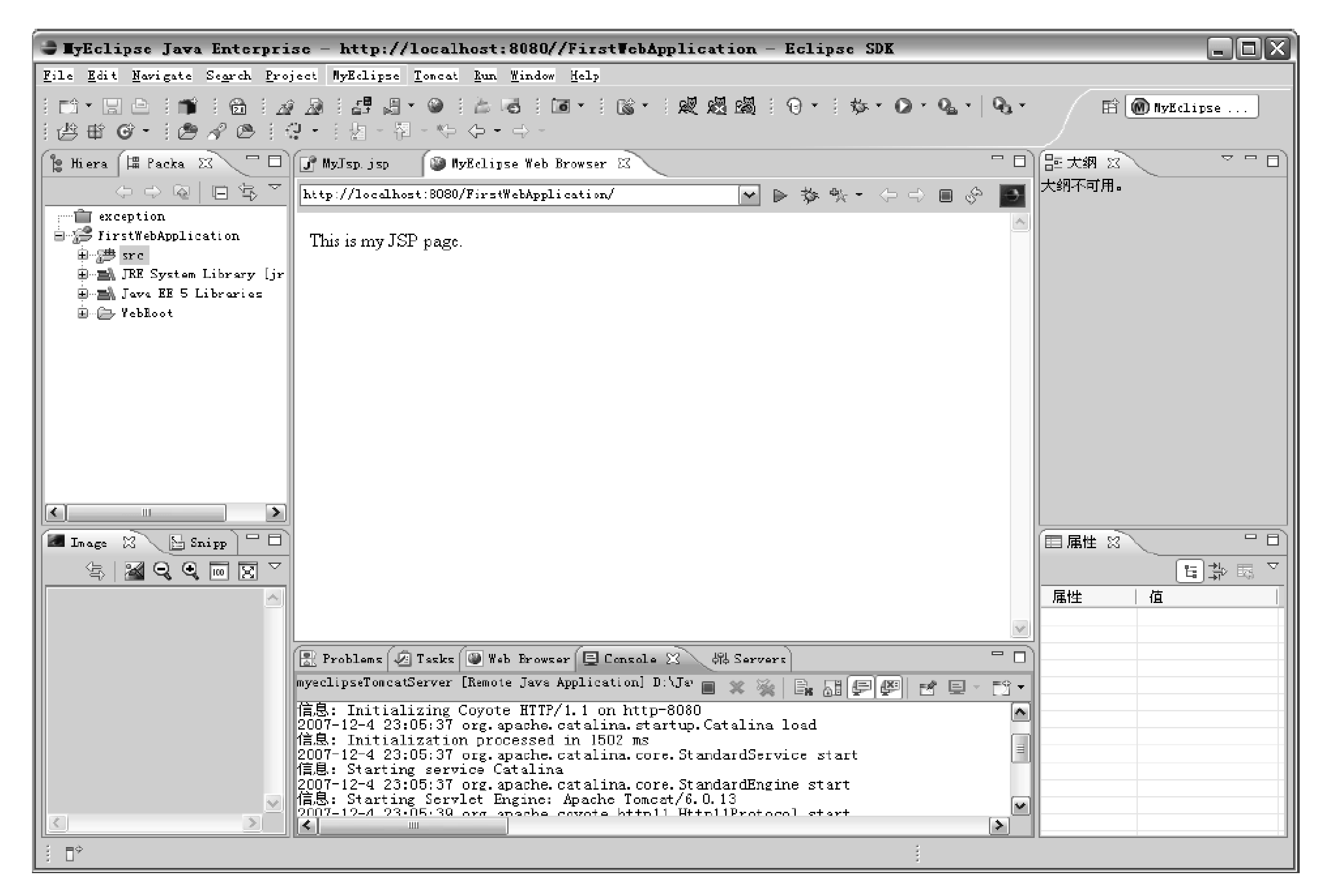

MyEclipse Java Enterprise - http://localhost:8080//FirstWebApplication - Eclipse SDK
File Edit Navigate Search Project MyEclipse Tomcat Run Window Help
MyEclipse ...
Hiera
Packa
exception
FirstWebApplication
src
JRE System Library [jr
Java EE 5 Libraries
WebRoot
MyJsp.jsp
MyEclipse Web Browser
http://localhost:8080/FirstWebApplication/
This is my JSP page.
大纲
大纲不可用。
Image
Snipp
属性
属性
值
Problems
Tasks
Web Browser
Console
Servers
myeclipseTomcatServer [Remote Java Application] D:\Ja
信息: Initializing Coyote HTTP/1.1 on http-8080
2007-12-4 23:05:37 org.apache.catalina.startup.Catalina load
信息: Initialization processed in 1502 ms
2007-12-4 23:05:37 org.apache.catalina.core.StandardService start
信息: Starting service Catalina
2007-12-4 23:05:37 org.apache.catalina.core.StandardEngine start
信息: Starting Servlet Engine: Apache Tomcat/6.0.13

图 A-9　运行 JSP 文件

高等学校计算机专业教材精选

计算机硬件

单片机嵌入式应用的在线开发方法　邵贝贝	ISBN 978-7-302-09658-0

计算机基础

大学计算机应用基础　陈良银	ISBN 978-7-302-17497-4
计算机科学导论教程　黄思曾	ISBN 978-7-302-15234-7
计算机应用基础教程(第 2 版)　刘旸	ISBN 978-7-302-15604-8

计算机原理

PC 系列机汇编语言程序设计　张虹	即将出版
操作系统原理教程(第 2 版)　孟静	即将出版
计算机系统结构　李文兵	ISBN 978-7-302-17126-3
计算机组成原理(第三版)　李文兵	ISBN 978-7-302-13546-3
微型计算机操作系统基础——基于 Linux/i386　任哲	即将出版
微型计算机原理与接口技术应用　陈光军	ISBN 978-7-302-16940-6

软件工程

软件工程实用教程　范立南	即将出版

数理基础

离散数学及其应用　周忠荣	ISBN 978-7-302-16574-3

算法与程序设计

C 语言程序设计教程　覃俊	ISBN 978-7-302-16903-1
C 语言上机实践指导与水平测试　刘恩海	ISBN 978-7-302-15734-2
Java 程序设计教程　孙燮华	ISBN 978-7-302-16104-2
Visual Basic 上机实践指导与水平测试　郭迎春	ISBN 978-7-302-15199-9
计算机程序设计经典题解　杨克昌	ISBN 978-7-302-163589
数据结构　冯俊	ISBN 978-7-302-15603-1
数据结构与算法(C++ 版)　游洪跃	ISBN 978-7-302-17502-5
数据结构与算法(C++ 版)实验和课程设计教程　游洪跃	ISBN 978-7-302-17503-2
算法实践与问题求解　孙广中	即将出版

数据库

数据库原理与应用案例教程　郑玲利	即将出版

图形图像与多媒体技术

AutoCAD 2008 中文版机械设计标准实例教程　蒋晓	ISBN 978-7-302-16941-3
计算机图形学基础教程(Visual C++ 版)　孔令德	ISBN 978-7-302-17082-2
计算机图形学实践教程(Visual C++ 版)　孔令德	ISBN 978-7-302-17148-5

网络与通信技术

Web 开发技术实用教程　陈轶	ISBN 978-7-302-17435-6
Web 开发技术实验指导　陈轶	即将出版
Web 数据库编程与应用　魏善沛	ISBN 978-7-302-17398-4
Web 数据库系统开发教程　文振焜	ISBN 978-7-302-15759-5
计算机网络技术与实验　王建平	ISBN 978-7-302-15214-9
计算机网络原理与通信技术　陈善广	ISBN 978-7-302-15173-9

读者意见反馈

亲爱的读者：

感谢您一直以来对清华版计算机教材的支持和爱护。为了今后为您提供更优秀的教材，请您抽出宝贵的时间来填写下面的意见反馈表，以便我们更好地对本教材做进一步改进。同时如果您在使用本教材的过程中遇到了什么问题，或者有什么好的建议，也请您来信告诉我们。

地址：北京市海淀区双清路学研大厦 A 座 602 室 计算机与信息分社营销室 收
邮编：100084　　　　电子邮件：jsjjc@tup.tsinghua.edu.cn
电话：010-62770175-4608/4409　　　　邮购电话：010-62786544

教材名称：Web 开发技术实用教程
ISBN：978-7-302-17435-6

个人资料

姓名：________ 年龄：______ 所在院校/专业：________

文化程度：________ 通信地址：________________

联系电话：________ 电子信箱：________________

您使用本书是作为：□指定教材 □选用教材 □辅导教材 □自学教材

您对本书封面设计的满意度：

□很满意 □满意 □一般 □不满意　改进建议________________

您对本书印刷质量的满意度：

□很满意 □满意 □一般 □不满意　改进建议________________

您对本书的总体满意度：

从语言质量角度看 □很满意 □满意 □一般 □不满意

从科技含量角度看 □很满意 □满意 □一般 □不满意

本书最令您满意的是：

□指导明确 □内容充实 □讲解详尽 □实例丰富

您认为本书在哪些地方应进行修改？（可附页）

____

____

您希望本书在哪些方面进行改进？（可附页）

____

____

电子教案支持

敬爱的教师：

为了配合本课程的教学需要，本教材配有配套的电子教案（素材），有需求的教师可以与我们联系，我们将向使用本教材进行教学的教师免费赠送电子教案（素材），希望有助于教学活动的开展。相关信息请拨打电话 010-62776969 或发送电子邮件至 jsjjc@tup.tsinghua.edu.cn 咨询，也可以到清华大学出版社主页（http://www.tup.com.cn 或 http://www.tup.tsinghua.edu.cn）上查询。